December 1977

The DEC EDITOR, EDT, is a text editor that is designed for use on many different DIGITAL operating systems. The DEC EDITOR allows you to create and update files. This manual describes the commands and features EDT uses.

DEC EDITOR
Reference Manual

Order No. AA-5789A-TC

SUPERSESSION INFORMATION:	This is a new manual.
OPERATING SYSTEMS AND VERSIONS:	RSTS/E Version 6C RSX-11M Version 3.1
SOFTWARE VERSION:	1.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, December 1977

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

PREFACE

This manual describes the DEC EDITOR, EDT, which is designed to run on many operating systems. The editor allows you to create files and manipulate their contents.

This manual contains an overview of the editor and its capabilities, but it is not written as a tutorial. This manual is a reference tool and contains documentation of all of EDT's features.

Several EDT commands – EXIT, INCLUDE, PRINT, RESTORE, SAVE, and WRITE – support options that require you to supply a name for either an input or an output file. In each case this document specifies that you supply only a filename. According to the requirements of your operating system, you may be required to enter a complete file specification which may include device specifications, filename, filetype or extension, and version number. However, you also have the option of entering a minimum file specification that is acceptable by your system.

In any case before you use EDT to create or access a file, you should consult your *System Reference Manual* or *User's Guide* for detailed instructions about file specifications, including system default values for unspecified filename components.

The /SEQuence and /UNsequenced command options, when used with the EXIT and WRITE commands, operate in a system dependent manner. Refer to your *System Reference Manual* or *User's Guide* for information about how your system assigns line numbers to the lines in files created by EDT.

The following symbols, when used in this manual, have the indicated meanings:

RET	Denotes pressing the RETURN key.
<LF>	Denotes pressing the LINE FEED key, or indicates an ASCII line feed character.
[ ]	Brackets enclose optional parameters.
{ }	Braces enclose parameters of which one must be specified.

Other documentation conventions are:

1. Unless otherwise indicated, all lines of user input are terminated with the RETURN key.
2. In the examples used in this manual, system printout is shown in black, and user input is shown in red.
3. CTRL/Z denotes holding down the CTRL key while typing the Z key, as when using the SHIFT/key combination. The slash is shown merely to tie the two actions together. CTRL is also used with other keys (e.g., CTRL/U, which deletes the current line before it is transmitted). The use of the CTRL/key combination may print a ^ and the key typed, e.g., CTRL/A may echo ^A on the terminal.
4. When this manual refers to a disk file or an on-disk version of a file, it is understood that this includes files contained on disk packs, cartridge disks, and floppy disks. It does not include files contained on magnetic tapes, cassettes, or paper tapes.

CHAPTER 1
INTRODUCTION TO THE DEC EDITOR

The DEC EDITOR, EDT, is a standard text editor offered on most DIGITAL operating systems. EDT features:

- English language-based commands
- Hard-copy and screen display
- On-line error diagnosis
- File and buffer I/O handling
- Line and Character Editing Modes
- Maneuverable cursor and line pointer
- A common editing language across systems

EDT editing functions include:

- Creation and manipulation of files and contents
- Complete backup facility
- On-line allocation of text storage buffers
- Definition and execution of groups of EDT commands
- Variable command range specifications
- Character string searches

1.1 SPECIFYING THE FILES YOU WANT TO EDIT

When you are ready to use EDT, you must issue a command from your system monitor to invoke EDT. Your system reference manual or user's guide describes the exact command that you must enter to invoke EDT. Once you have invoked EDT, you receive one of the following prompts at your terminal.

```
EDT>
  or
FILE?
```

These prompts indicate that you may enter input and/or output files that EDT will use during the editing session. Enter your input and/or output files in the following format:

```
EDT>[output file specification=] [input file specification]
```

1.2 BUFFERS AND FILES

When you invoke EDT, EDT creates a text buffer, or storage area, called MAIN. If you specify the name of an existing file as an input file when you invoke EDT, EDT copies the contents of that file into the MAIN text buffer. However, if you specify the name of a new file, EDT creates MAIN with no data stored in it. EDT performs the editing commands you issue on the contents of the text buffer. EDT commands cannot alter files, including the input file you specify when invoking EDT. EDT can, however, create a new file or version of a file to contain the edits you make. Commands can add to or take away from the contents of a buffer, move text from one buffer to another, or simply change EDT's position within a buffer.

EDT can create text buffers other than MAIN. For example, you can access other permanent files using the INCLUDE command to copy their contents into an alternate text buffer. EDT allows you to name the alternate text buffers you use, and to make edits to the buffer contents in the same way you edit the MAIN text buffer.

When you finish using EDT, you can use two commands to terminate EDT and return control to your operating system. The QUIT command erases the contents of all text buffers you used since invoking EDT. The EXIT command saves the contents of only the MAIN text buffer by writing MAIN's contents to the specified output file.

1.3 TEXT BUFFERS

Text buffers are temporary, on-line storage areas. EDT copies files into text buffers, inserts input from terminals into text buffers, and performs all EDT commands on the contents of text buffers. There is no limit to the size of text buffers other than the limits of the system that you are using; however EDT does not assign line numbers above 65535.

1.3.1 MAIN Text Buffer

The text buffer MAIN is the initial text buffer created by EDT. If you specify an existing file as the input file when you invoke EDT, EDT copies that file into MAIN. However if you specify a nonexistent file as the input file, EDT creates MAIN without any text in it.

MAIN is the only text buffer whose contents are written to the output file when you terminate EDT with an EXIT command.

1.3.2 Alternate Text Buffers

Alternate text buffers are temporary storage areas created by EDT as a result of commands you issue.

EDT creates a text buffer when you give it a name and then use the buffer name as part of a command. Buffer names can be up to six letters or numbers long. Each time you refer to a buffer, you must precede the buffer name with either an = (equal sign) or a %BUF notation. Thus, the following are all legal buffer references:

```
%BUF ALT1
=ALT1
%BUF 10SNE1
=10SNE1
```

When your system manager installs the EDT text editor program on your system, he determines the number of text buffers that will be available during an editing session. You will always be able to use the MAIN buffer and at least two alternate buffers, but you should check with your system manager to find out the maximum number of buffers available on your system.

EDT does not save the contents of any alternate buffers when you terminate the editing session. You can save the contents of alternate text buffers by either using the WRITE command to directly create a file from the buffer, or by using the MOVE or COPY command to place the alternate buffer's contents in the MAIN text buffer.

1.4 FILES

EDT uses three types of files to allow you to copy files, to create files, and to backup your editing session. The types of files are:

• Input files	files that are copied from disk
• Output files	files that are created on disk
• Temporary files	files that are saved on disk

1.4.1 Input Files

Input files are disk files that EDT copies into a text buffer. There are two ways to copy input files using EDT. First, you can specify an input file in response to the EDT> or FILE? prompt. EDT then copies the contents of the input file you specify and automatically places the copy in the MAIN text buffer. Second, you may issue an INCLUDE command. When you issue an INCLUDE command, you specify the name of the file you want to copy and the text buffer you want its contents stored in.

Since EDT can only manipulate the contents of text buffers, you must use either or both of these methods to access text that is stored in permanent files.

1.4.2 Output Files

Output files are disk files that EDT creates from the contents of text buffers. There are two ways to create permanent files. First, you can specify an output file in response to the EDT> prompt received when you invoked EDT. When you subsequently issue an EXIT command to terminate EDT, EDT writes the contents of the MAIN buffer to the output file you specified. Second, you may issue a WRITE or PRINT command. When you issue either of these commands, EDT writes out the contents of the buffer you specify to a permanent file that you must name.

If you specify a single file to serve as both input and output file, EDT accesses the current version of the file and then creates a new version of the file when you EXIT to terminate EDT. If you do not specify a file that can be used as an output file, you cannot EXIT unless you use the /RENAME option. Otherwise, if you want to save the edits you have made, you must WRITE out the contents of the buffer to a file and then QUIT to terminate EDT.

1.4.3 Temporary Files

Temporary files are disk files that EDT creates to store and access backup information on your editing session. You should periodically backup your editing session by issuing a SAVE command. This protects you from losing the edits you have made in the event you incur a loss of your text buffer's contents.

When you issue a SAVE command, EDT copies the status of all files in use as well as the contents of all your text buffers into a disk file that you specify. Do not try to access this file directly. It can only be correctly accessed by EDT's RESTORE command in a subsequent editing session.

After you issue the SAVE command, you may still edit the text buffers that you copied to the temporary disk file, If later in the same editing session you want to again backup your edits, issue another SAVE command.

The temporary file created by the SAVE command can only be accessed and used by EDT when you issue the RESTORE command. To re-create the contents of the text buffers you saved, and to re-establish the status of the files in use, issue the RESTORE command only when:

1. You invoke EDT without an input or output file
2. All text buffers are empty

1.5 EDT COMMAND LEVEL AND EDITING MODES

You automatically enter EDT's Command Level when you provide EDT with the files you will be using during the editing session. EDT signifies the Command Level by printing an * (asterisk) prompt on the left-hand side of your terminal display. When the * prompt is present, you can enter any EDT command. When you issue most commands, EDT executes them and then returns to the Command Level upon completion. However, the REPLACE, INSERT, and CHANGE commands invoke special editing modes that you must terminate in order to return to the Command Level. Other commands, EXIT and QUIT, terminate EDT.

1.5.1 Character Mode

When you enter the CHANGE command at EDT's Command Level, you invoke Character Mode. Character Mode is an editor within the EDT editor. Character Mode allows users with video display terminals to scroll through a text buffer and to issue subcommands to make insertions, deletions, replacements, and substitutions. While editing in Character Mode, EDT can only display a line-by-line copy of the up-to-date contents of the buffer.

See Chapter 3 for a description of how to edit in Character Mode by issuing subcommands.

1.5.2 Insert Mode

When you enter an INSERT or a REPLACE command at Command Level you invoke EDT's Insert Mode. Insert Mode allows you to enter characters at your terminal which are then placed in the text buffer. The INSERT command immediately invokes Insert Mode; the REPLACE command deletes some text before it invokes Insert Mode.

Once you enter Insert Mode, EDT does not prompt you to begin typing input. You should type the command line, follow it with a carriage return, and then begin typing the text you want to put into the buffer. When you have finished entering text, enter a line that contains only CTRL/Z. CTRL/Z returns EDT to Command Level. The following example illustrates the REPLACE command procedure as it appears on the terminal:

```
*REPLACE 40:60(RET)
WHEN IN THE COURSE(RET)
OF HUMAN EVENTS(RET)
IT BECOMES NECESSARY(RET)
FOR ONE PEOPLE TO(RET)
DISSOLVE THE POLITICAL(RET)
<CTRL/Z>
*
```

In the above example, EDT deletes the lines from 40 through 60, inclusive, and then inserts the five lines entered in the example in their place. Follow the same procedure when you enter the INSERT command. Refer to the INSERT and REPLACE commands in Chapter 3 for more detailed descriptions of their operation and specification.

1.6 LINE POINTER

The line pointer is an internal EDT mechanism that keeps track of EDT's position within text buffers. As you issue commands, EDT moves from line to line and from buffer to buffer and uses the line pointer to keep track of its position as it moves. The line pointer does not point at any part of the line; rather it indicates an entire line. The current position of the line pointer (the current line) can be displayed by typing a carriage return in response to the Command Level asterisk prompt. The position of the line pointer changes as EDT executes commands. Each command's effect on the line pointer is documented in the detailed command descriptions in Chapter 2.

1.7 LINE NUMBERS

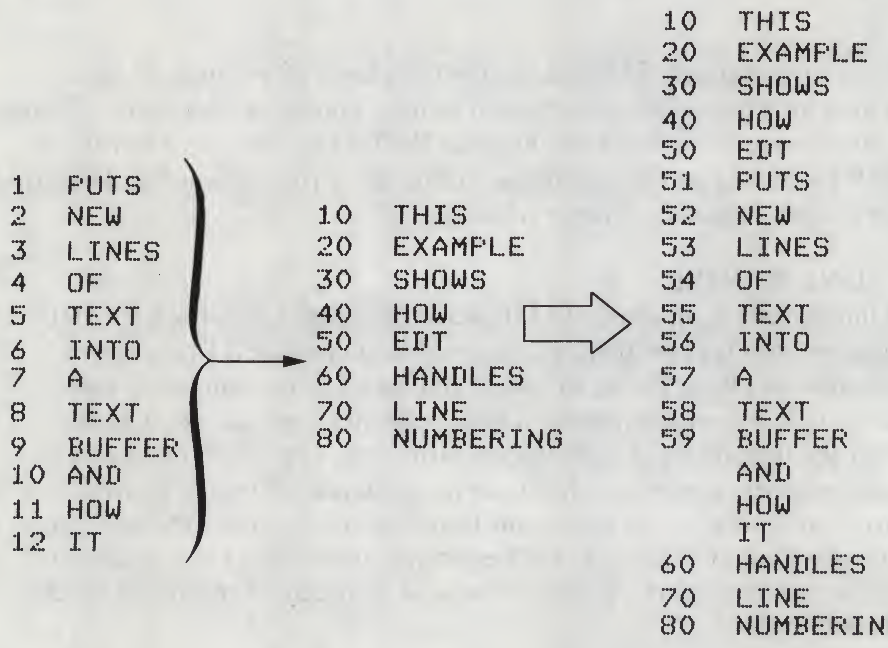
EDT automatically assigns a line number to each line in a text buffer. The default system line numbers are 10, 20, 30, 40, etc., and help you locate and reference the lines in your buffer. Line number are integers in ascending order from 1 to 65535. They appear on the left margin of the terminal display, separated from the text by a tab. Line numbers can be changed or completely removed from the buffer but are not part of the text. The RESEQUENCE command lets you specify the line numbers you want assigned.

Many commands that transpose, insert, or delete lines have an option, /SEQUENCE, that allows you to specify numbers for the transferred or inserted text. Often the line numbers that result after one of these commands have varying increments between them, and some lines may not have numbers at all. If this occurs, you can use the RESEQUENCE command to re-assign uniform line number increments to all the lines in your buffer.

1.7.1 Line Number Sequencing

As stated above in Section 1.7, some command actions change the line numbering of the text buffer and some commands put lines into the buffer. When EDT assigns numbers to the new lines, the new line numbers often do not conform to the numbers in the remainder of the buffer. Some commands take lines out of the buffer which also cause sporadic numbering of lines.

For example, the MOVE, REPLACE, INCLUDE and INSERT commands put lines into text buffers. If the lines in the buffer are numbered 10, 20, 30, etc. and 12 lines are inserted between lines 50 and 60, EDT assigns line numbers to the group as shown in Figure 1-1.



```
1  PUTS
2  NEW
3  LINES
4  OF
5  TEXT
6  INTO
7  A
8  TEXT
9  BUFFER
10 AND
11 HOW
12 IT
```

```
10  THIS
20  EXAMPLE
30  SHOWS
40  HOW
50  EDT
60  HANDLES
70  LINE
80  NUMBERING
```

```
10  THIS
20  EXAMPLE
30  SHOWS
40  HOW
50  EDT
51  PUTS
52  NEW
53  LINES
54  OF
55  TEXT
56  INTO
57  A
58  TEXT
59  BUFFER
    AND
    HOW
    IT
60  HANDLES
70  LINE
80  NUMBERING
```

Lines Being Inserted | Original Buffer | Resulting Buffer

Figure 1-1 Input Line Number Sequencing

As Figure 1-1 shows, EDT numbers as many of the new lines as possible without either re-assigning or changing any line numbers already in use. EDT then puts the remaining lines in their proper position without assigning them any numbers.

On the other hand, the MOVE, REPLACE and DELETE commands take lines out of text buffers. When you use these commands, EDT removes the lines you specify, as well as their line numbers, from the buffer. If a line number is no longer assigned to a line, it is available for either EDT or you to assign to a new line or to re-assign to an existing line.

1.8 TEXT AND STRING HANDLING

Line numbers are not the only way for you to locate and reference the text in buffers. You can direct EDT to search the buffer contents for a specific word or combination of letters, called a string. When you specify such a character string, you give EDT an object string to search for, and EDT examines the text buffer looking for an equivalent or match string. In its search, EDT first examines the current line to see if it contains a match of the object string. If no match is found in the current line, EDT then examines the next line, and the next, and so on until it either finds a match or reaches the end of the buffer.

Once you determine the object string and the action you want to perform, you must:

- Specify the string
- Dictate the direction of the search
- Determine what constitutes a match

1.8.1 String and Search Specification

EDT treats characters enclosed in quotation marks as object strings. If you specify a character or group of characters enclosed in ' ' (single quotation marks) or " " (double quotation marks) you direct EDT to search for the first occurrence of an identical string of characters (without the quotation marks) contained in the text buffer. " " and ' ' are the only character string identifiers that EDT accepts as part of a range specification.

The way you specify the object string determines the direction of EDT's search. If you enter "ABC", EDT searches from the current line toward the bottom of the buffer looking for the first occurrence of ABC. If you precede the object string with a - (minus sign), i.e., - "ABC", EDT searches from the current line toward the top of the buffer looking for the first occurrence of ABC.

1.8.1.1 SUBSTITUTE Command String Usage – The SUBSTITUTE command also works by locating character strings in the buffer. SUBSTITUTE commands have a different command syntax, and their character strings are usually delineated by / (slash) instead of ' or ". The / may be replaced by any printing non-alphanumeric character, e.g., @, <, #, >, etc. The basic SUBSTITUTE command format is:

```
S/object string/ replacement string/
```

where the first occurrence of the object string is located within the buffer and then changed to the specified replacement string. For example, S/ABC/XYZ/ locates the first occurrence of ABC in the buffer, and changes the characters ABC to XYZ. You can also issue the same command in the form S#ABC#XYZ#.

1.8.2 String Match Control

EDT lets you specify whether or not a match occurs when the case of the characters in the match string differs from the case of the characters in the object string. For example, if you are searching your text buffer for an occurrence of "BASIC", do you want EDT to return a match when it encounters the string "basic"?

The SET command allows you to specify whether match strings must correspond to the cases specified in the object string. If you enter the SET EXACT CASE command, EDT returns as a match only those strings that are identical in case and content to the object string. If you enter the SET EXACT NONE command, EDT returns any occurrence of the characters in the object string as a match, regardless of case. See the SET command description in Chapter 4 of this manual for a more detailed description.

1.9 EDT ERROR REPORTING

EDT has a special way of reporting any errors that you may encounter during its use. When EDT cannot perform a command, it prints a message on your terminal. This message explains why EDT cannot perform the command you specified. If the message does not fully explain the problem, you may request additional information by typing a ? (question mark) in response to the asterisk prompt that immediately follows the message. You may continue requesting information until EDT cannot display any additional messages. In most cases, the error should be explained after one or two messages. If the messages do not explain the error, consult the documentation that describes the command you are trying to execute.

In the following example, the user attempts to locate the next line of text that contains XYZ. Instead of typing "XYZ" or 'XYZ', the user incorrectly types "XYZ'. EDT displays the message "Missing string quote" to explain the error that prohibits it from locating the string. The uer then requests more information by typing question marks until the question marks do not return any more diagnostic information.

```
*"XYZ'
Missing string quote
*?
          A legal string must be surrounded by a pair of
          matching quote marks.
*?
          A quoted string consists of one or more characters
          delimited by " or '. For instance, "ABC" or 'DEF'.
*?
*
```

CHAPTER 2

EDT COMMANDS AND COMMAND SPECIFICATIONS

2.1 EDT COMMAND OVERVIEW

EDT makes changes to the text in response to commands that you enter at your terminal. You can issue commands only when you receive the asterisk prompt at Command Level.

When you enter a command, you have to specify the action you want EDT to take, and you have to specify the buffer and buffer portion you want to affect.

This chapter describes the components of the command string, and how you specify a command string. The final section of this chapter lists each command that you can enter, and contains detailed information about each command.

2.2 COMMAND STRING COMPONENTS

The EDT commands that you enter at Command Level consist of several parts. The entire command string can consist of:

- Command name
- Range specification
- Options
- Carriage Return

The command name tells EDT which command to implement. The range specification tells EDT which buffer or buffer portion is to be affected by the command. The options let you control the command execution or specify actions for EDT to perform during execution.

In one command or another, all of the above command string components are optional except the carriage return. EDT takes no action on any command string until it is followed by a carriage return.

2.2.1 Command Name

The command name that you specify in the command string tells EDT which command to execute. You may issue a specific abbreviation in place of the full command name, but only certain abbreviations are allowed. In the following list of EDT commands, the uppercase letters are the only acceptable abbreviation. A brief description of each command action is included.

• Change	invokes Character Mode
• COpy	copies lines from one location to another

• Delete	removes lines from the text buffer
• EXit	terminates EDT
• Find	repositions the line pointer
• INClude	copies a file into a text buffer
• Insert	accepts terminal input and places it in a text buffer
• Move	transfers lines
• PRint	generates a file that shows the EDT line numbers assigned to the lines you print out
• QUIT	terminates EDT
• Replace	deletes lines and then accepts and inserts replacement lines into the text buffer
• RESequence	renumbers the lines in the text buffer
• RESTore	re-creates all text buffer contents from a file created by the SAVE command
• SAve	creates a file that contains the contents of all text buffers in use at the time
• SEt	specifies display and match criteria
• SHow	displays current match and display criteria and buffer status
• Substitute	changes characters within lines of the text buffer
• Type	displays lines of the buffer on your terminal
• WRite	creates a file from text buffer contents
• Xeq	allows you to create and execute EDT commands as a group

2.2.2 Range Specification

The range specification that you enter in the command string tells EDT which lines and buffer are to be affected by the command. There are four basic types of range specifications:

- Single line ranges specify one line of text
- Variable line ranges specify an indeterminate number of lines of text
- Compound ranges specify a known number of lines of text
- Inclusive ranges specify criteria that determine which lines of text are included in the range

2.2.2.1 Single Line Range Specifiers – Single line range specifiers identify one line of a text buffer as the range specification. Single line range specifiers and the lines they specify are listed below in Table 2-1.

Table 2-1
Single Line Range Specifiers

Range Specifier	Meaning
nn	Line number nn
. (period)	The current line
"object string" or 'object string'	The first line located that contains an acceptable match of the object string
- "object string" or - 'object string'	The first preceding line that contains an acceptable match of the object string
%BE	The first line of the current text buffer
%E	The last line of the current text buffer
%L	The line in a previous text buffer at which the line pointer was positioned when the command to enter the current buffer was issued

2.2.2.2 **Variable Line Range Specifiers** – Variable line range specifiers identify an indeterminate number of lines in the text buffer as the range specification. Variable line range specifiers and the lines they identify are listed in Table 2-2.

Table 2-2
Variable Line Range Specifiers

Range Specifier	Meaning
=buffer name or %BUF buffer name	All lines contained in the text buffer "buffer name"
%BEF	All lines in the current text buffer from the first line in the buffer through the current line
%R	All lines in the current text buffer from the current line through the last line in the buffer
%WH	All lines in the current text buffer

2.2.2.3 **Compound Range Specifiers** – Compound range specifiers identify a specific number of lines in the buffer as a range specification. Compound range specifiers consist of single line range specifiers, range operators, and integers. Table 2-3 below lists the forms that compound range specifiers may take.

In the specifier column, "sr" represents a single range specifier, any "i" represents an integer. The range operators are :(colon) ;(semi-colon) ,(comma) +(plus sign) and -(minus sign).

Table 2-3
Compound Range Specifiers

Range Specifier	Meaning
sr1:sr2 or sr1 %THRU sr2	The lines between and including the first single line range (sr1) and the second single line range (sr2) where sr1 must precede sr2. For example, "ABC": "DEF" refers to all lines between the first line that contains ABC and the first line that contains DEF, inclusive.
sr;i or sr %FOR i	The total number of lines specified by the integer (i) that begins with the line specified by the single line range (sr). For example, 100;10 refers to the ten lines that follow line number 100.
sr,sr,sr, . . . or sr %AND sr %AND sr . . .	The lines identified by the single line ranges. For example, 10, 70, 130, 190 refers to only those lines numbered 10, 70, 130, and 190.
sr+i	The single line that is i lines after the specified single line range. For example, "ABC"+10 refers to the tenth line following the first line that contains an acceptable match of the string ABC.
sr-i	The single line that is i lines before the single line range. For example, "ABC"- 10 identifies the tenth line before the first line containing a match of ABC.

2.2.2.4 Inclusive Range Specifiers – The inclusive range specifiers identify the lines of text that are affected by a command by setting criteria that lines must meet. Only those lines that satisfy the criteria are used as the range specification. Generally, there are two types of inclusive range specifiers. The first is a combination of the variable line range buffer identifier and any single, compound, or variable range specifier. The second allows you to include all lines that contain a specific character string. These two types of inclusive range specifiers may also be used together to determine a single range. Table 2-4 illustrates inclusive range specifiers.

In the Range Specifier column, "rs" represents any legal range specification except a buffer name and string refers to an object string.

Table 2-4
Inclusive Range Specifiers

Range Specifier	Meaning
=buffer name rs or %BUF buffer name rs	The lines specified by any legal range specifier that are contained in the specified buffer. For example, =A "ABC":40 identifies the lines between the first line that contains an acceptable match of ABC and line number 40, inclusive, that are contained in text buffer A.
%ALL 'string'	All lines in the current text buffer that contain acceptable matches of the specified object string. For example, %ALL "ABC" identifies all lines in the current buffer that contain ABC.
=buffer name rs %ALL 'string'	All lines that contain acceptable matches of the object string within the range specification rs that are contained in the specified text buffer. For example, =A 10:100 %ALL 'ABC' identifies all lines that contain ABC that are between lines 10 and 100 inclusive in text buffer A.

2.2.3 Options

Command line options either allow you to control the command execution or specify actions that EDT is to take when the execution has been completed. Each detailed command description in this chapter includes a description of each option that is allowed with the command. The following table shows options, their functions, and the commands they are used with. Acceptable abbreviations of options include the leading / (slash) and the upper case characters that follow it.

Table 2-5
Command Options

Option	Meaning	Commands
/BRief	Displays first ten characters of lines in range	Type and Substitute
/FIle	Specifies a disk file that is either accessed or created	WRite, SAve, INClude, PRint and RESTore
/NL	Allows deletion of line terminators	Change
/Query	Prompts you to control execution of command	Substitute, Move, COpy and Delete
/REname	Specifies an output file	EXit
/SEQuence	Specifies sequence numbers	EXit, INClude, Insert, COpy, Move, Replace, RESequence, WRite
/- Type	Inhibits display of lines affected by the command	Substitute
/UNsequenced	Inhibits renumbering of lines	EXit, INClude, Insert, COpy, Move, Replace, RESequence, and WRite

2.3 DETAILED DESCRIPTIONS OF EDT COMMANDS

The rest of this chapter lists EDT commands alphabetically and gives detailed instruction in their formats and uses. Information concerning the function, format, operation, options and usage of each command is contained in each command description. A summary of the format and function of each command is contained in Appendix A.

CHANGE

Use the CHANGE command to invoke EDT's Character Editing Mode. The CHANGE command takes the format:

```
Change [range] [/NL]
```

where:

range
specifies the lines that you will be able to access while in Character Mode. If you do not specify a range, the entire contents of the current text buffer can be accessed.

/NL
specifies that, while in Character Mode, you can delete line terminators, e.g., form feeds, line feeds or other line terminators.

USING THE CHANGE COMMAND

When You Can Use the CHANGE Command

You can use the CHANGE command only if you are using a video display terminal.

You must use the SET TERMINAL command to correctly establish your terminal type before you can accurately edit in Character Mode.

Using CHANGE with a Range

While in Character Mode, you only can edit the contents of the specified range. If you want to access lines that are outside the range, you first must issue an EX subcommand and then re-enter the CHANGE command specifying the new range.

Using the /NL Option

When you invoke Character Mode and specify the /NL option, EDT treats end-of-line characters as single characters that may be inserted or deleted. EDT displays these characters as follows:

<FF>	indicates an ASCII form feed
<LF>	indicates an ASCII line feed
<VT>	indicates an ASCII vertical tab character

When you delete an end-of-line character, EDT concatenates the following line to the line whose end-of-line character was deleted.

If you attempt to delete a line terminator without having specified the /NL option, EDT takes no action in response to the subcommand you enter.

Terminating Character Editing Mode

Typing EX returns EDT to Command Level from Character Mode. Typing EX does not terminate EDT if issued from Character Mode.

Typing QUIT terminates both Character Mode and EDT. QUIT does not create or update any files.

COPY

Use the COPY command to transfer lines from one location to another while keeping them in their original location. The COPY command takes the format:

```
COpy range-1 %TO range-2     [/Query]
                             [/SEQuence:initial number:increment]
                             [/UNsequenced]
```

where:

range-1
specifies the lines that are to be copied. If you specify more than one line in range-1, all the lines you specify are copied.

range-2
specifies the line ahead of which the lines in range-1 are copied. If you specify more than one line in range-2, the lines in range-1 are copied ahead of the first line in range-2.

/Query
allows you to specify how each line in range-1 is to be treated. Before EDT transfers each line, it prints the line and waits until you type one of the following responses:

Y or YES	copies the line.
N or NO	does not copy the line.
Q or QUIT	stops copying lines and returns to Command Level.
A or ALL	copies the remaining lines in the range without printing them first.

/SEQuence:initial number:increment
assigns specific line numbers to the copied lines. The ":initial number" argument specifies the number of the first line that was copied, and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates the copied lines with line numbers of 100, 200, 300, 400, etc.

/UNsequenced
causes EDT to copy the lines without assigning them line numbers.

USING THE COPY COMMAND

Line Pointer Position After the COPY Operation

After a line or group of lines has been copied, EDT positions the line pointer at the first line of the copied lines in their new position preceding the first line in range-2.

Specifying Ranges

You cannot overlap ranges. Range-1 and range-2 cannot contain any of the same lines.

Default Line Numbering Scheme

If you do not use either the /SEQuence or /UNsequenced option to specify line numbers, EDT attempts to number the new lines in the following manner:

- If EDT can number the new lines by incrementing the existing line numbers in steps of 10, it does so as long as the resulting line numbers do not duplicate line numbers or violate the ascending order of line numbers.
- If incrementing by 10 would result in an illegal line number, EDT begins incrementing subsequent new lines in steps of 1.
- If incrementing the new lines in steps of 1 would result in illegal or duplicate line numbers, EDT places all further lines in the correct order, but does not assign them any line numbers.

Examples

1. CO .:".MAC" %TO =X 40
   This command copies all lines, beginning with the current line and continuing through the first line containing .MAC, to a position preceding line 40 in buffer X.
2. CO .;20 %TO %BE
   This command copies 20 lines, starting with the current line, to the beginning of the current text buffer.

DELETE

Use the DELETE command to delete lines from a text buffer. DELETE takes the following format:

```
Delete [range] [/Query]
```

where:

range
specifies a range of lines to be deleted. If you do not specify a range, the current line is deleted.

/Query
specifies that you want to control the DELETE operation using EDT prompts and responses. EDT prompts you by displaying the line to be deleted; you can then enter any one of the following responses:

Y or YES	delete the prompt line.
N or NO	do not delete the prompt line.
Q or QUIT	stop deleting lines and return to Command Level.
A or ALL	delete all remaining lines in the range; do not display the deleted lines.

USING THE DELETE COMMAND

Position of Line Pointer After the DELETE Operation
After a line or a group of lines has been deleted, EDT positions the line pointer at the line immediately following the last line deleted.

Line Pointer Position After the DELETE /Query Operation
The /Query option allows you to delete specific lines at random from the specified range of lines. After you delete lines using the /Query option, EDT positions the line pointer at the undeleted line nearest the beginning of the range.

Using Line Numbers to Specify a Range
If you enter nonexistent line numbers in a range specification, EDT rejects the specification and displays an error message.

Examples

1. D 20:40
   Deletes lines 20 through 40, inclusive, from the text buffer.
2. D %ALL 'ABC'/Q
   Finds all the lines in the buffer containing ABC, and after displaying each line, waits for you to enter a Y, N, Q, or A response.
3. D =BUF2 %ALL '438'
   Deletes all lines containing 438 from text buffer BUF2.

EXIT

Use the EXIT command to terminate EDT, return control to the system monitor, and to save the contents of the MAIN text buffer. EXIT takes the following format:

```
         [/REname:filename]
    EXit [/SEQuence:initial number:increment]
         [/UNsequenced]
```

where:

/REname:filename
specifies the output file to which the contents of the MAIN text buffer are written. The /REname option overrides the output file specified when EDT was invoked.

/SEQuence:initial number:increment
specifies the line numbers that EDT assigns to the lines in the output file. The ":initial number" argument specifies the number of the first line in the file, and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates a file with line numbers of 100, 200, 300, 400, etc.

Specifying /SEQuence without arguments causes EDT to assign the default line numbers 10, 20, 30, etc. to the output file.

/UNsequenced
causes EDT to create the file without assigning it line numbers.

USING THE EXIT COMMAND

Affecting Text Buffers

The EXIT command only saves the contents of the text buffer MAIN. The contents of all other text buffers are lost if they are not incorporated into MAIN or written to permanent files through the WRITE command.

Using the /REname Option

The /REname option allows you to specify an output file whether or not an output file was originally specified in the EDT invocation command.

Example

1. EX/RE:SYS.MAC/SEQ:2:2
   EDT writes the contents of the MAIN text buffer to the file SYS.MAC. The lines in SYS.MAC are numbered 2, 4, 6, 8, 10, etc.

FIND

Use the FIND command to move the line pointer. The FIND command takes the following format:

```
    Find range
```

where:

range
specifies the line to which the line pointer is positioned. If you specify a range of more than one line, EDT positions the line pointer at the first line in the range.

USING THE FIND COMMAND

Terminal Output
The FIND command does not generate any terminal output.

Examples

1. F =X 20
   EDT positions the line pointer at line number 20 in text buffer X.
2. F- 'ABC'
   EDT searches upward and positions the line pointer at the first line encountered that contains ABC.
3. F 20:40
   EDT positions the line pointer at line 20.

INCLUDE

Use the INCLUDE command to locate a storage file and to copy it into a text buffer. The INCLUDE command takes the following formats:

```
                    /FI:filename
INClude [range]     [/SEQ:initial number:increment]
                    [/UNsequenced]
```

where:

range
specifies the line ahead of which the contents of the file are to be inserted. If you specify a range of more than one line, EDT inserts the file ahead of the first line in the range. If you do not specify a range, EDT inserts the file ahead of the current line.

/FI:filename
specifies the filename of the file that is to be copied.

/SEQuence:initial number:increment
assigns specific line numbers to the included lines. The ":initial number" argument specifies the number assigned to the first line, and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates the new lines with line numbers of 100, 200, 300, 400, etc.

/UNsequenced
causes EDT to include the lines without assigning line numbers.

USING THE INCLUDE COMMAND

Line Pointer Position After the INCLUDE Operation

After a disk file has been included into a text buffer, EDT positions the line pointer at the last line of the newly included text.

Using the INCLUDE Command with Text Buffers

The INCLUDE command copies a file into your current text buffer or to an alternate text buffer, allowing access to any portion of that file during your editing session. Once the file is copied, you can use other EDT commands to manipulate its contents.

For example, to access a portion of a file other than the one you are currently editing, perform the following steps:

1. Use the INCLUDE command to copy the file into an alternate text buffer.
2. From that buffer, use the MOVE command to transfer the desired portion of the file to the MAIN buffer.

Using the INCLUDE Command with Line Numbers

If more lines exist in the file being included than there are line numbers available, EDT correctly inserts the extra lines into the text buffer but does not assign them any line numbers.

If you specify line numbers with the /SEQuence option that conflict with other line numbers in the file, the new lines are included without line numbers.

If you do not use either the /SEQuence or /UNsequenced option to specify line numbers, EDT attempts to number the new lines in the following manner:

- If EDT can number the new lines by incrementing the existing line numbers in steps of 10, it does so as long as the resulting line numbers do not duplicate line numbers or violate the ascending order of line numbers.
- If incrementing by 10 would result in an illegal line number, EDT begins incrementing subsequent new lines in steps of 1.
- If incrementing the new lines in steps of 1 would result in illegal or duplicate line numbers, EDT places all further lines in the correct order, but does not assign them any line numbers.

When you use the INCLUDE command to insert a file into a text buffer that already contains some data, you may want to use the RESEQUENCE command to establish uniform line number increments throughout that buffer.

Examples

1. INC =ADD /FI:SYS.MAC
   EDT copies the contents of file SYS.MAC into text buffer ADD. If ADD contains some data, EDT inserts the new lines at the beginning of the buffer.
2. INC 'ABC' /SEQ:100:2/FI:EDT.FTN
   EDT first copies the contents of file EDT.FTN, then inserts them before the next line in the current buffer that contains the character string ABC, and finally numbers the new lines as 100, 102, 104, 106, etc.

INSERT

Use the INSERT command to place text you type at your terminal into a text buffer. The INSERT command takes the following format:

```
Insert [range]  [/SEQuence:initial number:increment]
                [/UNsequenced]
```

where:

range
specifies the line ahead of which the text you type is to be inserted. If you specify more than one line in the range, the text is inserted ahead of the first line in the range. If you do not specify a range, the text is inserted ahead of the current line.

/SEQuence:initial number:increment
assigns specific line numbers to the newly inserted lines. The ":initial number" argument specifies the number assigned to the first new line and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates new lines with line numbers of 100, 200, 300, 400, etc.

/UNsequenced
causes EDT to insert the new lines without assigning them line numbers.

USING THE INSERT COMMAND

Line Pointer Position After an INSERT Operation
After a line or group of lines has been inserted into a text buffer, EDT positions the line pointer at the last inserted line.

Entering Text from the Terminal
When you issue an INSERT command, EDT first positions the line pointer ahead of the first line in the range. EDT then goes into Insert Mode and begins accepting lines of text you enter at the terminal. When all the lines have been entered from the terminal, terminate Insert Mode by typing the <CTRL/Z> character. EDT inserts the lines you typed ahead of the first line in the specified range.

NOTE

When you issue an INSERT command, EDT enters Insert Mode but does not prompt you for input. Type the command line and a carriage return, and then the lines you want to insert.

Inserting Text with Line Numbers

If you specify line numbers with the /SEQuence option that conflict with other line numbers in the file, EDT inserts the new lines without line numbers.

If you do not use either the /SEQuence or /UNsequenced option to specify line numbers, EDT attempts to number the new lines in the following manner:

- If EDT can number the new lines by incrementing the existing line numbers in steps of 10, it does so as long as the resulting line numbers do not duplicate line numbers or violate the ascending order of line numbers.
- If incrementing by 10 would result in an illegal line number, EDT begins incrementing subsequent new lines in steps of 1.
- If incrementing the new lines in steps of 1 would result in illegal or duplicate line numbers, EDT places all further lines in the correct order, but does not assign them any line numbers.

If you insert a large number of lines, you may want to enter a RESEQUENCE command to uniformly renumber the lines in the buffer.

Terminating Insert Mode

After you type all the lines to be inserted, type a line that contains only CTRL/Z. This CTRL/Z returns EDT to Command Level where you may enter additional commands.

Examples

1. I =B 40

   EDT goes into Insert Mode, and inserts all the lines you enter ahead of line 40 in buffer B.
2. I ".OR"

   EDT inserts the lines you enter ahead of the next line encountered that contains .OR.
3.
   ```
   *I (RET)
   LINE 1 (RET)
   LINE 2 (RET)
   LINE 3 (RET)
   <CTRL/Z>
   *
   ```

   The example above illustrates the entire INSERT command sequence. The command, I, which is issued while EDT is at Command Level, causes EDT to enter Insert Mode. EDT then accepts the three lines of input from the terminal and puts them ahead of the current line. Finally, CTRL/Z returns EDT to Command Level.

MOVE

Use the MOVE command to transfer lines from one location to another and to delete them from their original location. The MOVE command takes the format:

```
Move range-1 %TO range-2  [/Query]
                          [/SEQuence:initial number:increment]
                          [/UNsequenced]
```

where:

range-1
specifies the lines that are to be moved. If you specify more than one line in range-1, all the lines you specify are moved.

range-2
specifies the line ahead of which the lines in range-1 are moved. If you specify more than one line in range-2, the lines in range-1 are moved ahead of the first line in range-2.

/Query
allows you to specify how each line in range-1 is to be treated. Before EDT transfers each line, it prints the line and waits until you type any one of the following responses:

Y or YES	moves the line.
N or NO	does not move the line.
Q or QUIT	stops moving lines and returns to Edit mode.
A or ALL	moves the remaining lines in the range without printing them first.

/SEQuence:initial number:increment
assigns specific line numbers to the transferred lines. The ":initial number" argument specifies the number of the first line that was moved, and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates the transferred lines with line numbers of 100, 200, 300, 400 etc.

/UNsequenced
causes EDT to transfer the lines without assigning them line numbers.

USING THE MOVE COMMAND

Line Pointer Position After a MOVE Operation

After a line or group of lines has been moved, EDT positions the line pointer at the first line of the transferred lines in their new position preceding the first line in range-2.

Specifying Ranges
You cannot overlap ranges. Range-1 and range-2 cannot contain any of the same lines.

Using the MOVE Command with Line Numbers
If you use the MOVE command to transfer a large number of lines, you can issue a RESEQUENCE command to establish uniform line number increments.

If you do not use either the /SEQuence or /UNsequenced option to specify line numbers, EDT attempts to number the new lines in the following manner:

- If EDT can number the new lines by incrementing the existing line numbers in steps of 10, it does so as long as the resulting line numbers do not duplicate line numbers or violate the ascending order of line numbers.
- If incrementing by 10 would result in an illegal line number, EDT begins incrementing subsequent new lines in steps of 1.
- If incrementing the new lines in steps of 1 would result in illegal or duplicate line numbers, EDT places all further lines in the correct order, but does not assign them any line numbers.

Examples

1. M .:".MAC" %TO =X 40
   This command moves all lines, beginning with the current line and continuing through the first line containing .MAC, to a position preceding line 40 in buffer X.
2. M .;20 %TO %BE
   This command moves 20 lines, starting with the current line, to the beginning of the current text buffer.
3. M =X %TO =MAIN %E
   This example appends the contents of text buffer X to the end of the MAIN buffer.

PRINT

Use the PRINT command to create a file from the contents of a text buffer. The file that you create by using the PRINT command contains as part of the text the EDT line numbers assigned to the lines in the range. The PRINT command takes the following format:

```
PRint [range] /FI:filename
```

where:

range
specifies the buffer contents that you want printed in the permanent storage file you create.

If you do not specify a range, EDT uses the entire current text buffer as the range.

/FI:filename
specifies the name of the output file that you create. You must specify the filename using the /FI option.

USING THE PRINT COMMAND

Line Pointer Position After a PRINT Operation

EDT does not reposition the line pointer after the PRINT command generates an output file.

Effect on Buffers

If you do not specify a range, the PRINT command generates the file with the whole of the current text buffer as its contents.

When to Use the PRINT Command

Use the PRINT command to create a file that contains as part of the text the EDT line numbers that are assigned to the lines in the range you specify.

Examples

1. PR =B 10:70/FI:INT.FOR
   This command creates a file INT.FOR on disk containing lines 10 through 70, inclusive, of buffer B.
2. PR =MAIN/FI:SY.MAC
   This command creates a file, SY.MAC, from the contents of the MAIN text buffer. The file SY.MAC has as part of the text of the file the line numbers that existed in the MAIN buffer at the time the PRINT command was issued.

QUIT

Use the QUIT command to terminate EDT and to return control to the system monitor without saving the contents of any text buffers. The QUIT command takes the following format:

```
QUIT
```

USING THE QUIT COMMAND

How QUIT Affects Buffers

The QUIT command does not modify any buffers. It does not write out the contents of the MAIN buffer or any other buffer, nor does it generate any files.

How QUIT Affects Files

If you invoke EDT with only an input file, you must either issue a QUIT command to return to the system monitor or issue an EXIT command with the /RENAME option to specify an output file. You may also use the WRITE command to generate a disk file that contains those edits.

If you use the WRITE command to generate a disk file before you issue the QUIT command, that file is not affected by QUIT or any other subsequent EDT command.

When to Use the QUIT Command

The QUIT command is especially useful if you have made an error that results in the loss of all or part of a buffer you are editing. If you make a mistake that results in a loss of your buffer's contents, issuing a QUIT command negates all changes made since EDT was invoked.

Example

1. QUIT
   This command erases all text buffer contents and returns control to the system monitor.

REPLACE

Use the REPLACE command to delete and insert one or more lines of text in a text buffer. The REPLACE command takes the format:

```
Replace [range]  [/SEQuence:initial number:increment]
                 [/UNsequenced]
```

where:

range
specifies those lines that are to be deleted. The replacement lines begin with the first line in the range.

/SEQuence:initial number:increment
assigns specific line numbers to the replacement lines. The ":initial number" argument specifies the number of the first line and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates replacement lines with line numbers of 100, 200, 300, 400, etc.

/UNsequenced
causes EDT to create the replacement lines without assigning them line numbers.

USING THE REPLACE COMMAND

Line Pointer Position After a REPLACE Operation

After a line or group of lines has been replaced, EDT positions the line pointer at the last replacement line.

Using REPLACE Without a Range

If you do not specify a range, the REPLACE command deletes the current line and inserts the replacement lines in its place.

Using REPLACE with Line Numbers

If you replace a single line with several lines, you can issue a RESEQUENCE command to assign uniform line number increments to all lines in the text buffer.

If you specify line numbers with the /SEQuence option that conflict with other line numbers in the file, EDT inserts the replacement lines without line numbers.

Default Line Numbering Scheme

If you do not use either the /SEQuence or /UNsequenced option to specify line numbers, EDT attempts to number the new lines in the following manner:

- If EDT can number the new lines by incrementing the existing line numbers in steps of 10, it does so as long as the resulting line numbers do not duplicate line numbers or violate the ascending order of line numbers.
- If incrementing by 10 would result in an illegal line number, EDT begins incrementing subsequent new lines in steps of 1.
- If incrementing the new lines in steps of 1 would result in illegal or duplicate line numbers, EDT places all further lines in the correct order, but does not assign them any line numbers.

Entering Replacement Text from the Terminal
When you issue the REPLACE command, EDT enters Insert Mode but does not prompt you to begin entering the replacement lines of text. You should type the REPLACE command line, a carriage return, and then type the lines you want to insert.

After you type the replacement lines, type a line that contains only CTRL/Z. CTRL/Z returns EDT to Command Level where you may issue further commands.

Examples

1. R RET
   GOTO 40 RET
   <CTRL/Z>
   This example deletes the current line, replaces it with "GOTO 40" and returns to Command Level.
2. R 'ABC'+20 RET
   IF XYZ THEN DEF RET
   <CTRL/Z>
   EDT deletes the twentieth line after the first occurrence of ABC and replaces it with the "IF . . . THEN . . ." statement before returning to Command Level.

RESEQUENCE

Use the RESEQUENCE command to assign new line numbers to the lines in text buffers. The RESEQUENCE command takes the following format:

```
RESequence [range]   [/SEQuence:initial number:increment]
                     [/UNsequenced]
```

where:

range
specifies the lines and text buffer to be renumbered. If you do not specify a range, EDT resequences all lines in the current text buffer.

/SEQuence:initial number:increment
assigns specific line numbers to the lines in the range. The ":initial number" argument specifies the number of the first line in the range, and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 renumbers the range with line numbers of 100, 200, 300, 400, etc.

/UNsequenced
causes EDT to remove line numbers from the range.

USING THE RESEQUENCE COMMAND

Line Pointer Position After a RESEQUENCE Operation

After a buffer or a group of lines has been resequenced, EDT positions the line pointer at the first line that was resequenced.

Specifying a Range with RESEQUENCE

If you do not specify a range, EDT renumbers the entire current text buffer.

As a general practice, resequence the entire text buffer instead of a portion of it. This ensures that uniform line number increments exist for your convenience.

Using RESEQUENCE with Line Numbers

If you do not specify either the /SEQuence or /UNsequenced option, EDT assigns the system default line numbers 10, 20, 30, etc. to the lines in the text buffer.

When you renumber only part of the text buffer, none of the line numbers you assign can match those in the part of the buffer not being renumbered. If you specify a line number for the resequenced range that is less than the line numbers before it, all lines in the range are written without numbers.

The highest sequence number that you can use is 65535; the lowest is 1.

When to Use the RESEQUENCE Command

After you insert or delete a large number of lines, the line numbers of the remaining lines may be difficult to work with. Use the RESEQUENCE command to restore uniform line number increments to the text buffer.

Displaying Line Numbers

Use the TYPE command to display on your terminal the line numbers that are currently assigned to the contents of the buffer. Use the PRINT command to create a file that contains the new line numbers as part of the text.

Examples

1. RES
   EDT renumbers the current text buffer's lines as 10, 20, 30, etc.
2. RES =A /SEQ:1:1
   EDT renumbers the lines of buffer A as 1, 2, 3, 4, etc.

RESTORE

Use the RESTORE command to first locate a file created by a SAVE command and to then use the file to re-create the status of all files and the contents of all text buffers as they were preserved in the file. The RESTORE command takes the following format:

```
RESTore /FI:filename
```

where:

/FI:filename
specifies the filename of the storage file that was created by previously issuing a SAVE command. That file, when accessed by the RESTORE command, re-creates the contents of all text buffers in use at the time the SAVE command was entered.

USING THE RESTORE COMMAND

In order to issue a RESTORE command to re-create all text buffer contents preserved by a SAVE command it must be the first command that you issue. Therefore you must:

1. Invoke EDT without specifying either an input or output file.
2. Have all text buffers empty.
3. Specify as the RESTORE command input file that file created by a SAVE command in a previous editing session.

Line Pointer Position After a RESTORE Operation

After your text buffer contents have been restored, EDT positions the line pointer at the line that was the current line when the SAVE command created the input file.

Examples

1. REST /FI:TXT.SAV
   This command must be the first command you issue after invoking EDT without specifying either an input or an output file. EDT then locates the file TXT.SAV, and uses that file to re-create the contents of all text buffers in use at the time the SAVE command created TXT.SAV.

SAVE

Use the SAVE command to preserve the contents of the text buffers and status of all files you use during your editing session in a file which you specify. The SAVE command takes the following format:

```
SAve /FI:filename
```

where:

/FI:filename
specifies the name of the output file that you create. The file you create contains the current status of all the buffers in use. You must use the /FI option to specify the name of the file you create.

USING THE SAVE COMMAND

Line Pointer Position After a SAVE Operation
EDT does not reposition the line pointer after the SAVE command generates the output file.

Effect on Buffers
The SAVE command does not alter the contents of the text buffers in use when it creates a disk file.

Examples

1. SA /FI:TXT.SAV
   This command saves all text buffer contents by creating the file TXT.SAVon disk. The file TXT.SAV contains the contents of all the text buffers in use at the time the SAVE command was issued. You may access the TXT.SAV file only by issuing a RESTORE command to re-create the contents of the text buffers.

SUBSTITUTE

Use the SUBSTITUTE command to change characters within lines of the text buffer. The SUBSTITUTE command takes the following formats:

```
                                                      [/BRief]
Substitute/object string/replacement string/[range]   [/Query]
                                                      [/-Type]

    or

Substitute Next
```

where:

object string
specifies the characters in the text buffer that you want to locate and to change. See Section 1.8.1.1 for more information about object strings.

replacement string
specifies the characters with which you want to replace the object string characters. See Section 1.8.1.1 for more information about replacement strings.

range
specifies the range of lines within which all occurrences of the object string are changed to the replacement string. If you do not specify a range, EDT changes only the first occurrence of the object string that it locates.

/BRief
specifies that EDT display the first ten characters of each line in which it makes a substitution. An additional form of the /BRief command is supported:

```
/BRief:n
```

This form of the /BRief option specifies that EDT display the first n characters of each line in which it makes a substitution.

/Query
specifies that you want to control the operation by using EDT prompts and responses. EDT prompts you by displaying the substitution about to be made and waits for you to enter any one of the following responses:

Y or YES	makes the substitution.
N or NO	does not make the substitution.
Q or QUIT	stops substituting and returns to Command Level.
A or ALL	makes substitutions in the rest of the appropriate lines in the range without printing them first.

/-Type
inhibits EDT's automatic display of each line after a substitution is made in that line:

Next
repeats the operation of the SUBSTITUTE command that it must immediately follow.

USING THE SUBSTITUTE COMMAND

SUBSTITUTE Command Operation

The SUBSTITUTE command changes every occurrence of the object string to the replacement string within a specified range. If you do not specify a range, only the next occurrence of the object string is changed. EDT's search for the object string begins with the current line.

EDT displays each line that contains a substitution on the terminal after it makes the substitution. If you specify a range that requires substitutions in more lines than you wish to have displayed, specify the /-Type option to inhibit the display.

SUBSTITUTE NEXT Operation

The S N or SUBSTITUTE NEXT command format must immediately follow another SUBSTITUTE command. When entered, S N repeats a single substitution as performed by the command it follows.

The S N command format does not use a range or any options. It allows you to perform one more substitution that has already been defined without requiring you to re-type the command string.

Line Pointer Position After a SUBSTITUTE Operation

After EDT makes substitutions, it positions the line pointer at the last line in which a substitution was made.

Using String Delimiters

You can replace the /(slash) string delimiter with any printing character except a letter or number, i.e., ?(question mark) ;(semicolon) '(apostrophe) etc.

You can omit the final string delimiter if no range or options follow and if the omitted delimiter is the last character in the command line.

Examples

1. S/ABC/XYZ
   EDT changes the next occurrence of ABC to XYZ. If ABC exists on the current line, it is changed. If not, the contents of the buffer are searched, and the first occurrence is changed.
2. S'/'\'
   EDT changes the first occurrence of / (slash) to \ (backslash), and uses the ' (apostrophe) as the string delimiter.

3. S/A/B/.
   Changes every occurrence of A to B in the current line only.

   Note that the final / (slash) string delimiter must be included here since the current line range, . (period), is specified.
4. S/001/100/%WH/Q
   EDT attempts to change every occurrence of 001 to 100 in the entire text buffer, but queries you before making each substitution.
5. S/.AND/.OR/%REST/-T
   EDT changes every occurrence of .AND to .OR from the current line through the end of the text buffer but does not display any of the resulting lines.
6. S/.MAC/.BAS//-T
   S N
   EDT changes the first occurrence of .MAC to .BAS, but does not display the changed line. The S N command then changes the next occurrence of .MAC to .BAS and automatically displays the line.

   Note in the first command line that two back-to-back / (slashes) are used. The first of these indicates the end of the replacement string and the second indicates that an option follows.

TYPE

Use the TYPE command to display lines of text at your terminal. The TYPE command takes the formats:

[Type] range

or

Type [range] [/BRief]

where:

range
specifies the lines that you want displayed at your terminal.

/BRief
specifies that EDT display only the first ten characters of each line in the range. An additional form of the /BRief option is supported:

/BRief:n

This alternate form displays the first n characters of each line in the range.

USING THE TYPE COMMAND
You may issue a command that consists of only a range. The contents of that range are displayed on the terminal. If you issue this format of the TYPE command, you cannot include any command options.

Line Pointer Position After a TYPE Operation
After EDT displays the lines in the range on your terminal, it positions the line pointer at the first line in the range.

Examples

1. 'ABC':"DEF"
   This command displays all lines of text from the first line that contains ABC through the first line that contains DEF, inclusive.
2. T 10:100/BR:5
   This command displays the first five characters of each line with line numbers between 10 and 100, inclusive.
3. %WH
   This command displays the entire text buffer contents.

WRITE

Use the WRITE command to create a file from the contents of a text buffer. The WRITE command takes the following format:

```
                     /FI:filename
     WRite [range]   [/SEQuence:initial number:increment]
                     [/UNsequenced]
```

where:

range
specifies the buffer contents that you want to store in the file you create.

/FI:filename
specifies the name of the output file that you create. You must use the /FI option to specify the name of your output file.

/SEQuence:initial number:increment
assigns specific line numbers to the output file. The ":initial number" argument specifies the number of the first line in the file, and the ":increment" argument specifies the increment between numbers. For example, /SEQ:100:100 creates a file with line numbers of 100, 200, 300, 400, etc.

Specifying /SEQuence without arguments causes EDT to use the current line numbers.

/UNsequenced
causes EDT to create the file without assigning line numbers.

USING THE WRITE COMMAND

Line Pointer Position After a WRITE Operation

EDT does not reposition the line pointer after it generates the output file.

Effect on Buffers

The WRITE command does not alter the contents of the range or buffer when it creates a permanent file.

Using WRITE Without a Range

If you do not specify a range, the WRITE command generates the output file with the whole of the current text buffer as its contents.

When to Use the WRITE Command

Use the WRITE command to generate a file that contains the contents of alternate text buffers that you want to save. When you enter the EXIT command, EDT saves only the contents of the MAIN text buffer; all other buffers are erased.

Use the WRITE command to save the contents of the MAIN buffer if you did not specify an output file when you invoked EDT.

Examples

1. WR =B 10:70/FI:INT.FOR
   This command creates a file INT.FOR on disk containing lines 10 through 70, inclusive, of buffer B.
2. WR =MAIN /UNS/FI:SY.MAC
   This command creates a file, SY.MAC, from the contents of the MAIN text buffer. SY.MAC has no line numbers assigned to it.

CHAPTER 3
CHARACTER EDITING MODE

3.1 CHARACTER MODE OVERVIEW

When you enter the CHANGE command at EDT's Command Level, you invoke Character Mode. Character Mode is an editor within the EDT editor. It allows users with video display terminals to scroll through a text buffer and to issue special subcommands to make insertions, deletions, replacements, and substitutions.

While editing in Character Mode, EDT can only display an up-to-date, line-by-line copy of the buffer's contents. EDT updates the terminal display so that the screen always incorporates any changes you make. When you delete a line from the buffer, EDT removes that line from the display. Thus, if a character, word, or line appears out of order on your screen, it is out of order in the text buffer as well.

3.2 CURSOR

While in Character Mode, EDT uses a cursor instead of a line pointer to identify its current position within the buffer. The cursor is a visible representation of EDT's position that can be moved within the buffer and can be positioned at any character in the buffer.

If your terminal has them, you may use special keypad characters that look like arrows to manipulate the cursor. Pressing the down arrow moves the cursor to the first character of the next line and displays that line if it is not already displayed. Repeatedly pressing the down arrow will cause additional lines to appear on your screen if it can display multiple lines. Other arrows move the cursor up a line at a time, or left or right across lines.

Section 3.7 contains more information about the keypad arrows and alternate ways to move the cursor. These alternate methods of cursor movement use portions of the subcommand lines described below in their operation.

3.3 CHARACTER MODE STRING USAGE

Character Mode uses character strings in much the same way as the commands you issue at Command Level. When you want to locate a character string, enclose it in quotation marks just as if you were specifying a Command Level object string. While a Command Level search locates the line that contains the match string, a Character Mode search locates the characters that make up the match string and positions the cursor at the first character in the string.

3.4 USING INSERT MODE WITHIN CHARACTER MODE

Insert Mode works in Character Mode almost exactly as it does with Command Level commands. When you enter an I subcommand, EDT invokes Insert Mode and inserts the characters you type on your terminal into the text buffer at the location identified by the cursor. When you enter the R subcommands, EDT deletes some text before it invokes Insert Mode. As in Command Level usage, Insert Mode does not display a prompt and you should begin typing the text you want to insert immediately after you issue the subcommand.

Character Mode's usage of Insert Mode differs from Command Level's only in the way it terminates Insert Mode. While using Insert Mode at Command Level, you must terminate Insert Mode by issuing the CTRL/Z character on a line by itself. When terminating Insert Mode from Character Mode, enter the CTRL/Z character immediately following the last character you want inserted. If you type a carriage return to allow you to enter the CTRL/Z on its own line, EDT inserts the carriage return into the buffer as an input character.

3.5 SUBCOMMAND STRING COMPONENTS

The subcommand string that you enter in Character Mode consists of several parts. Subcommand strings can consist of:

- Subcommand name
- Repetition count
- String type
- Carriage return

In one subcommand string or another, all of these components can be omitted except the carriage return. EDT takes no action on a subcommand string until it is followed by a carriage return.

A subcommand string cannot contain any blanks. To issue a subcommand to delete 5 words, type D5W. If you type D 5 W, you will receive the error message: Illegal Subcommand.

3.5.1 Subcommand Names

The subcommand name that you specify tells EDT which editing function to execute. Subcommand names take the form of abbreviations only. The following list shows the only acceptable form of the subcommand names. A brief description of their actions is included.

D	deletes text from the buffer
EX	terminates Character Editing Mode and returns to EDT Command Level
I	inserts text into the buffer

QUIT terminates EDT in the same manner as the QUIT command – without modifying any files or including any edits you might have previously made

R replaces text in the buffer by first deleting and then inserting text

S performs character substitutions

3.5.2 Repetition Count

The repetition count is used in R and D subcommand strings, and in cursor movement subcommands. The repetition count is an integer, and tells EDT how many words, characters, or lines to delete from the buffer, or how many characters, words, or lines to move the cursor. If you do not include a repetition count in an R or D subcommand string, EDT assumes a count of 1.

3.5.3 String Types

The string type in an R or D subcommand or a cursor movement subcommand string tells EDT the text unit involved. String types may be one of the following:

- C indicates that characters are to be deleted or moved
- W indicates that words are to be deleted or moved
- L indicates that lines are to be deleted or moved

3.5.4 Substitution String Types

The string types used in an S subcommand string are the same as and follow the same rules as the object string and replacement string in the Command Level SUBSTITUTE command. The format for the Character Mode S subcommand is:

```
S/object string/replacement string/
```

As in the SUBSTITUTE command, you may replace the / (slash) delimiter with any printing, non-alphanumeric character.

3.6 SPECIFYING SUBCOMMANDS

This section contains reference information for each subcommand you can issue in Character Mode.

3.6.1 The D Subcommand

Use the D subcommand to delete characters from the text buffer. The D subcommand takes the format:

```
D [repetition count] string type
```

where:

repetition count
is an integer that tells EDT how many of the indicated string-types are to be deleted. If you do not specify a repetition count, EDT assumes a value of 1. The repetition count can be positive or negative.

string type
specifies the units that you want to delete. String type may be one of the following:

C	deletes characters
N	deletes words
L	deletes lines

Using the D Subcommand
The D subcommand begins by deleting the current character (that character under which the cursor is positioned). The total number of string types deleted is indicated by the repetition count you enter. If you enter a negative repetition count, you delete toward the beginning of the buffer. If you enter a positive repetition count, you delete toward the end of the buffer.

3.6.2 The EX Subcommand

Use the EX subcommand to terminate Character Mode and to return to Command Level. The EX subcommand takes no additional arguments.

3.6.3 The I Subcommand

Use the I subcommand to invoke Insert Mode to begin inserting text into the buffer. The text you type at your terminal is inserted into the buffer at the current character (that character under which the cursor is positioned).

To terminate Insert Mode and to return to Character Mode, type a CTRL/Z immediately following the last character you want inserted into the text buffer. The text you insert is automatically placed in the text buffer as shown in the screen display. You can insert as many characters, words, or lines of text as you need.

3.6.4 The QUIT Subcommand

Use the QUIT subcommand to terminate EDT directly from Character Mode. When you use the QUIT subcommand, EDT does not make any edits, does not update any buffers, and does not generate any output files.

3.6.5 The R Subcommand

Use the R subcommand to replace text in the buffer. The R subcommand takes the format:

```
R [repetition count] string type
```

where:

repetition count
is an integer that tells EDT how many of the indicated string types to delete from the buffer. If you do not specify repetition count, EDT assumes a value of 1. The repetition count may be positive or negative.

string type
specifies the units that you want to delete. String types may be one of the following:

C	deletes characters
W	deletes words
L	deletes lines

Using the R Subcommand
The R subcommand begins by deleting the text you specify with the repetition count and string type. This text is removed from your screen display. When text is no longer being removed, begin typing replacement text on your terminal. The text you type is inserted into the text buffer in the position displayed on your screen. When you have finished entering text, type CTRL/Z to return to Character Mode.

3.6.6 The S Subcommand

Use the S subcommand to make changes to characters in the buffer in the same way you use the SUBSTITUTE command. The S subcommand takes the format:

```
S/object string/replacement string/
```

where:

object string
specifies the characters in the buffer that you want to change.

replacement string
specifies the characters you want to substitute for the object string.

Using the S Subcommand
The S subcommand returns a match of the object string according to the criteria specified by the SET command which is described in Chapter 4. EDT begins its search for a match of the object string by looking at the character that follows the current character. When the change is made, the resulting text appears on your terminal.

3.7 CURSOR MANIPULATION

If you do not enter a subcommand name as part of a subcommand string, and instead enter only a repetition count and string type, EDT moves the cursor accordingly. For example, entering 20L moves the cursor to the beginning of the line that is twenty lines closer to the end of the buffer.

Entering - 6C moves the cursor six characters toward the beginning of the buffer.

Alternate methods of moving the cursor without changing text are listed below with the effects they have:

BL
moves the cursor to the beginning of the current line

BW
moves the cursor to the beginning of the current word

BR
moves the cursor to the first character in the range specified in the CHANGE command

EL
moves the cursor to the end of the current line

EW
moves the cursor to the end of the current word

ER
moves the cursor to the end of the range that was specified in the CHANGE command

“string” or
‘string’
moves the cursor to the next occurrence of the string in the buffer

- “string” or
- ‘string’
moves the cursor toward the top of the buffer until a match of the string is found

n“string” or
n‘string’
moves the cursor to the nth occurrence of the string in the buffer

- n“string” or
- n‘string’
moves the cursor toward the top of the buffer until the nth occurrence of the string is found

“ ” or
‘ ’
moves the cursor to the next occurrence of the last string you specified, and must directly follow the entry of an object string in the format “string” or ‘string’. The quotation marks may not have any spaces between them. If you type “ ”, you will locate the next blank in the buffer.

If your terminal is equipped with a special keypad, you may use the directional arrows to control the cursor movement. The arrows have the effects listed below.

↑ moves the cursor to the first character in the line above the current line

↓ moves the cursor to the first character in the line following the current line

→ moves the cursor one character to the right

← moves the cursor one character to the left

CHAPTER 4
OTHER COMMANDS

This chapter describes commands that control the operation of other commands or that execute a predetermined series of commands. The SET and SHOW commands described in this chapter cannot themselves change the text in the buffer.

SET

Use the SET command to establish criteria that are used by other EDT commands to flag uppercase or lowercase characters, and to establish the correct parameters for the terminal you use. The SET command takes the format:

```
SET  {  CAse        [ UPPER ]  }
     {              [ LOWER ]  }
     {              [ NONE  ]  }
     {                         }
     {  EXACT       [ CAse  ]  }
     {              [ NONE  ]  }
     {                         }
     {              [ HCPY  ]  }
     {              [ VT05  ]  }
     {              [ VT50  ]  }
     {  TErminal    [ VT52  ]  }
     {              [ VT55  ]  }
     {              [ VT61  ]  }
     {              [ LA30  ]  }
     {              [ LA36  ]  }
```

where:

CAse
specifies that the characters of the case you specify, either UPPER, LOWER, or NONE, are to be preceded by an apostrophe when they are displayed on the terminal. If you specify CASE UPPER, all uppercase characters are flagged. If you specify CASE LOWER, all lowercase characters are flagged. If you specify CASE NONE, no characters are flagged, and any previously entered CASE specifications are terminated. EDT does not flag characters while in Character Mode.

EXACT
specifies whether object string matches have to conform to the cases specified in the object string. If EXACT CASE is specified, match strings must conform to the cases of characters specified in the object string. If EXACT NONE is specified, only the characters must match, case matches are not required.

TErminal
specifies the terminal type that you are using. The valid terminal types are listed above in the command format. Enter the terminal type that corresponds to the terminal you are using.

USING THE SET COMMAND

Default Values of the SET Command
If you do not enter a SET command during your editing session, the following values are used:

CASE	NONE
EXACT	NONE
TERMINAL	no default value

Using the SET CASE Command
The SET CASE UPPER and SET CASE LOWER commands do not alter the characters in the text buffer; they only change the way the characters are displayed on the terminal.

You should generally use the SET CASE LOWER command when you have an uppercase only terminal. All characters that subsequently appear with a preceding apostrophe are lowercase characters. Lowercase characters cannot be changed or inserted from an uppercase only terminal, but they can be deleted.

EDT does not flag characters while in Character Mode. You cannot distinguish between uppercase and lowercase while in Character Mode if you have an all uppercase terminal.

Using the SET EXACT Command
The SET EXACT commands do not cause any changes in the text buffer. These commands only control the requirements that EDT sets for accepting a string match.

Using the SET TERMINAL Command
Setting the correct terminal type is mandatory only when you edit in Character Mode. Generally, however, you should always have the correct terminal type set.

Examples

1.
   ```
   *SET CASE LOWER (RET)
   *40 (RET)
   40    IF ABC 'L'T 'X'Y'Z GOTO 'X'X'X
   ```
   The text of line 40 actually exists in the buffer as:
   ```
   40    IF ABC lt xyz GOTO xxx
   ```
   This example illustrates that SET CASE LOWER displays all lowercase characters with a leading apostrophe.

2.
   ```
   *SET EXACT CASE (RET)
   *'ABC' (RET)
   NO SUCH LINE!
   *SET EXACT NONE (RET)
   *'ABC' (RET)
   130    IF abc, GOTO x
   ```

This example illustrates that, when SET EXACT CASE has been specified, EDT accepts only exact case matches.

SHOW

Use the SHOW command to display buffer and software version information and options established by the SET command. The SHOW command takes the following format:

```
SHOW  ┌BUffers  ┐
      │CAse     │
      │EXACT    │
      │TErminal │
      └VErsion  ┘
```

where:

BUffers
specifies that EDT display the name of each buffer you have used since invoking EDT, a summary of its contents, and indicate the current buffer with a leading = (equal sign).

CAse
specifies that EDT display the CASE option currently in effect as established by the SET command.

EXACT
specifies that EDT display the EXACT option currently in effect as established by the SET command.

TErminal
specifies that EDT display the TERMINAL option currently in effect as established by the SET command.

VErsion
specifies that EDT display the version number of the current EDT program.

USING THE SHOW COMMAND
The SHOW command, except for the BUFFERS and VERSION options, displays only those options that have been specified, or which can be specified, through the SET command.

Examples

1.
```
*SH CASE
UPPER
*
```
This example shows that the UPPER option of the SET command is in effect.

2.

```
*SH BU
MAIN  14   LINES
A     0    LINES
BUF2  143  LINES
=BUF3 90   LINES
*
```

This example displays the buffers referenced since EDT was invoked, the order in which they were created, and a summary of their contents. The current buffer is shown with a leading equal sign.

XEQ

Use the XEQ command to execute a previously entered sequence of EDT commands. The XEQ command takes the following format:

```
Xeq     range
```

where:

range
specifies the lines of text that contain the EDT commands to be executed.

USING THE XEQ COMMAND

When to Use XEQ

Use the XEQ command if you have identical or complex edits to perform, or if you have an established series of commands that are often issued.

Line Pointer Position After an XEQ Operation

EDT positions the line pointer as directed by the commands contained in the XEQ range.

How XEQ Affects the Range

The XEQ command cannot alter the contents of the XEQ range in any way.

How to Terminate Insert Mode if Invoked by XEQ

Terminating Insert Mode invoked during execution of an XEQ command must be done as follows:

> Create the sequence of commands that will be executed by the XEQ command, including the INSERT or REPLACE command that invokes Insert Mode.
>
> Enter the new or replacement lines as part of the sequence of commands contained in the range.
>
> When all new or replacement lines are entered, enter ^ (up-arrow) Z as part of the sequence of commands. When EDT executes the sequence of commands, the ^Z is interpreted as CTRL/Z.

Termination of XEQ

When the commands in the range have been executed, EDT returns to Command Level.

Examples

1. X 10:30

   where lines 10 through 30 contain:

   ```
   10      'ABC'
   20      S/40/70/.
   30      RET
   ```

   EDT searches for the next line that contains ABC, and attempts to change all occurrences of 40 to 70 within that line. The blank line implied by the RET symbol advances the line pointer past the now current line that contains ABC so that further X 10:30 commands locate the next succeeding line that contains ABC.

2. X =B

   where buffer B contains:

   ```
   =MAIN %BE
   'ABC'
   R
   XYZ
   ^Z
   ```

   This example returns the line pointer to the first line of the MAIN buffer, locates and deletes the first line that contains an occurrence of ABC, inserts a line consisting of XYZ, and terminates Insert Mode with the ^ (up-arrow) Z indicator.

APPENDIX A
SUMMARY OF EDT COMMANDS

Command Format	Description
Change [range] [/NL]	Invokes Character Editing Mode
COpy range-1 %TO range-2 [/Q] [/SEQ] [/UN]	Copies the lines in range-1 to a position ahead of the first line in range-2.
Delete [range] [/Query]	Deletes lines from the buffer.
EXit [/RE:filename.type] [/SEQuence] [/UNsequenced]	Terminates EDT; writes contents of MAIN text buffer to specified output file.
Find range	Moves the line pointer to the first line in the range.
INClude [range] /FI:filename.type [/SEQuence] [/UNsequenced]	Locates a file and copies it into a text buffer.
Insert [range] [/SEQuence] [/UNsequenced]	Invokes Insert Mode and places the text typed at the terminal in the buffer ahead of the first line in the range.
Move range-1 %TO range-2 [/Q] [/SEQ] [/UN]	Transfers the lines in range-1 to a position ahead of the first line in range-2.
PRint [range] /FI:filename	Generates an output file from the contents of the range. The output file contains EDT line numbers as part of the text.

Command Format	Description
QUIT	Terminates EDT; saves no edits or text buffers; generates no files.
Replace [range] [/SEQuence] [/UNsequenced]	First deletes the lines in the range, then invokes Insert Mode and places text typed at terminal in buffer in place of deleted text.
RESequence [range] [/SEQuence] [/UNsequenced]	Assigns new line numbers to the lines in the range.
RESTore /FI:filename	Locates specified file created by a SAVE command and uses the file's contents to restore the status and contents of the text buffers.
SAve /FI:filename	Creates a file that contains the status and contents of the text buffers currently in use.
UPPER SEt CAse LOWER NONE EXACT CAse NONE HCPY VT05 VT50 TErminal VT52 VT55 VT61 LA30 LA36	Establishes criteria that other EDT commands use in their operation; flags uppercase or lowercase characters and establishes proper terminal parameters.
SHow BUffers CAse EXACT TErminal VErsion	Displays the values established by the SET command as well as current buffer status and software version information.

Command Format	Description
Substitute/str-1/str-2/[range] [/BR] [/Q] [/-T]	Changes str-1 to str-2. If range is specified, changes all str-1s in the range. If no range, changes only the first str-1 encountered.
[Type] range or Type [range] [/BRief]	Displays the contents of the range on your terminal.
WRite [range] /FI:filename [/SEQuence] [/UNsequenced]	Creates an output file from the contents of the range.
Xeq range	Executes the EDT commands contained in the range.

INDEX

DEC EDITOR
Reference Manual
AA-5789A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

____
____
____
____
____
____
____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

____
____
____
____
____
____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

____
____
____
____
____
____

Please indicate the type of user/reader that you most nearly represent.

□ Assembly language programmer
□ Higher-level language programmer
□ Occasional programmer (experienced)
□ User with little programming experience
□ Student programmer
□ Non-programmer interested in computer concepts and capabilities

Name ____________________ Date ____________________

Organization ____

Street ____

City ____________________ State __________ Zip Code __________
or
Country

Please cut along this line.

-- **Fold Here** --

-- **Do Not Tear - Fold Here and Staple** --

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML 5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.

Introduction to RMS-11

Order No. AA-0001A-TC

digital equipment corporation · maynard. massachusetts

First Printing, January 1977

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

4/78-14

CONTENTS

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

This manual describes the concepts and facilities of Record Management Services for the PDP-11 (RMS-11).

RMS-11 is supported on a number of PDP-11 operating systems. The indexed file organization and associated facilities are extended features available with the RMS-11K product. The COBOL-11, BASIC-PLUS-2, and MACRO-11 languages provide access to RMS-11 features. This manual, therefore, is intended for users with varying operating system and language backgrounds. Since certain capabilities are available only to MACRO-11 programmers, you should consult the reference manual and user's guide associated with a particular language processor for detailed information on the use of RMS-11.

CHAPTER 1

INTRODUCTION

1.1 RMS-11 OVERVIEW

Record Management Services for the PDP-11 (RMS-11) is a set of general-purpose file-handling capabilities. In combination with a host operating system, it provides you with efficient and flexible facilities for data storage, retrieval, and modification. When writing programs in BASIC-PLUS-2, COBOL-11, or MACRO-11, you can select processing methods from among RMS-11's file structuring and accessing techniques that are best suited to a specific application.

To understand how RMS-11 provides this flexibility, you need to be aware of four RMS-11 concepts:

1. File organizations and access modes

2. File attributes

3. Program operations on RMS-11 files

4. The runtime environment

The manner in which RMS-11 builds a file is called its organization. RMS-11 provides three file organizations -- sequential, relative, and indexed. The organization of a file establishes the techniques you can use to retrieve and store data in the file. These techniques are known as access modes. The access modes that RMS-11 supports are 1) sequential, 2) random, and 3) record's file address (RFA).

You can use an application program or an RMS-11 utility, when you create an RMS-11 file, to specify the organization and characteristics of the file. These characteristics are known as the attributes of the file. Among the attributes you specify are storage medium, file and protection specifications, record format and size, and file allocation information.

After RMS-11 creates a file according to the file attributes you specify, application programs can store, retrieve, and modify data in it. These program operations can occur at the logical or physical level. At the logical level, an RMS-11 file is a collection of individual records. The record is the unit of information to which RMS-11 provides access. At the physical level, a file is a collection of units called virtual blocks. When bypassing the record processing capabilities of RMS-11, programs access these virtual blocks through a technique known as block I/O.

During runtime, RMS-11 and the host operating system provide an environment for user programs that permits file sharing and reduces the number of buffers required. When a program accesses files at the logical level, RMS-11 additionally supports 1) multiple access streams to a single file, 2) synchronous or asynchronous record operations, and 3) move and locate record transfer modes.

1.2 PRESENTATION OF INFORMATION IN THIS MANUAL

The presentation of information in this manual corresponds to these four RMS-11 concepts.

Chapter 2 describes the logical structure, or organization, of RMS-11 files and the access modes available with each file organization.

Chapter 3 details file attributes and their role in creating an RMS-11 file.

Chapter 4 summarizes program operations, at both the logical and physical level, that permit the retrieval, storage, and modification of data in files.

Chapter 5 is an overview of the runtime environment within which user programs process RMS-11 files.

CHAPTER 2

RMS-11 FILE ORGANIZATIONS AND ACCESS MODES

This chapter describes the logical structure and accessing capabilities of RMS-11 files.

A file is a collection of related information. Application requirements establish the nature of this information. For example, a company might maintain personnel information (employee names, addresses, job titles, and so forth) in one file and product information (part numbers, prices, specifications, and so forth) in a second, separate file. Within each of these files, the information is divided into records. In the personnel file, it would be logical for all the information on a single employee to constitute a single record and for the number of records in the file to equal the number of employees. Similarly, each record in the product information file would represent a description of a single product. Again, the number of records in the file reflects the requirements of a particular application -- in this case, a central registry of products sold by a company.

Each record in the personnel and product files would be subdivided into discrete pieces of information known as data fields. The user would define the number, location within the record, and logical interpretation of these data fields. Programmers at the company's data processing installation would write applications that always interpret a particular data field in records of the personnel file as the name of an employee. Likewise, they would interpret another data field, in records of the product file, as a part number. Figure 2-1 illustrates records that might occur in a personnel and a product file.

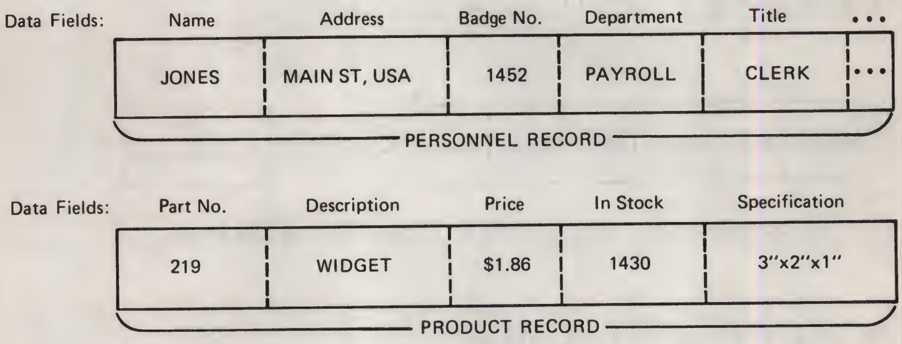


Figure 2-1 Personnel and Product Records

Thus, you can completely control the grouping of data fields into records and records into files. The relationship among data fields and records is known to you and is embedded in the logic of your programs. In contrast, RMS-11 does not require or employ an awareness of logical relationships among information in the files. Rather, RMS-11 processes records as single units of data. Your programs either build records and pass them to RMS-11 for storage in a file or issue requests for records while RMS-11 performs the necessary operations to retrieve the records from a file.

The purpose of RMS-11, then, is to ensure that every record written into a file can be subsequently retrieved and passed to a requesting program as a single logical unit of data. The structure, or organization, of a file establishes the manner in which RMS-11 stores and retrieves records. The way a program requests the storage or retrieval of records is known as the access mode. The access mode that can be used depends on the organization of a file. The sections of this chapter, therefore, describe:

- RMS-11 file organizations
- RMS-11 access modes
- Sample uses of RMS-11 file organizations.

2.1 RMS-11 FILE ORGANIZATIONS

When creating a file, you have a choice of three file organizations:

1. Sequential
2. Relative
3. Indexed

2.1.1 The Sequential File Organization

In the sequential file organization (see Figure 2-2), records appear in physical sequence. Each record, except the first, has another record preceding it, and each record, except the last, has another record following it. The physical order in which records appear is always identical to the order in which the records were originally written to the file by an application program.

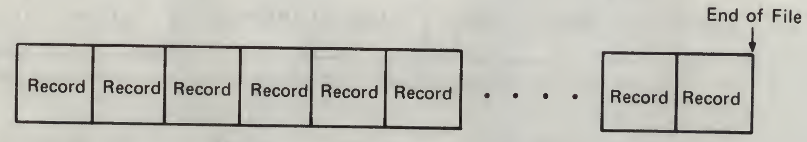


Figure 2-2 Sequential File Organization

2.1.2 The Relative File Organization

When you select the relative organization, RMS-11 structures a file as a series of fixed-size record cells. Cell size is based on the size you specify as the maximum permitted length for a record in the file. RMS-11 considers these cells as successively numbered from 1 (the first) to n (the last). A cell's number represents its location relative to the beginning of the file.

Each cell in a relative file can contain a single record. There is no requirement, however, that every cell contain a record. Empty cells can be interspersed among cells containing records.

Since cell numbers in a relative file are unique, you can use them to identify both a cell and the record, if any, occupying that cell. Thus, record number 1 occupies the first cell in the file, record number 17 occupies the seventeenth cell, and so forth. When you use a cell number to identify a record, it is also known as a relative record number. Figure 2-3 depicts the structure of a relatively organized file.

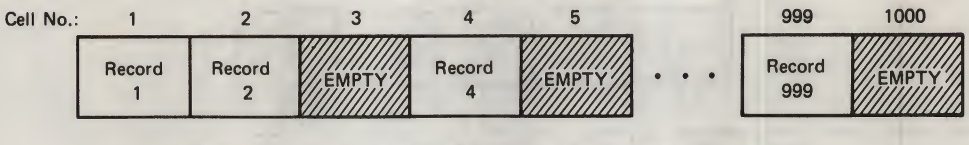


Figure 2-3 Relative File Organization

2.1.3 The Indexed File Organization[1]

Unlike the physical ordering of records in a sequential file or the relative positioning of records in a relative file, the location of records in the indexed file organization is transparent to your program. RMS-11 completely controls the placement of records in an indexed file. The presence of keys in the records of the file governs this placement.

A key is a character string present in every record of an indexed file. The location and length of this character string is identical in all records. When creating an indexed file, you decide which character string in the file's records is to be a key. By selecting such a character string, you indicate to RMS-11 that the contents (i.e., key value) of that string in any particular record written to the file can be used by a program to identify that record for subsequent retrieval.

You must define at least one key for an indexed file. This mandatory key is the primary key of the file. Optionally, you can define additional keys (i.e., alternate keys). Each alternate key represents an additional character string in records of the file. The key value in any one of these additional strings can also be used as a means of identifying the record for retrieval.

As programs write records into an indexed file, RMS-11 locates the values contained in the primary and alternate keys. From the values in keys within records, RMS-11 builds a tree-structured table known as an index. An index consists of a series of entries. Each entry contains a key value copied from a record that a program wrote into the file. With each key value is a pointer to the location in the file of the record from which the value was copied. RMS-11 builds and maintains a separate index for each key you define for the file. Each index is stored in the file. Thus, every indexed file contains at least one index -- the primary key index. When you define alternate keys, RMS-11 builds and stores an additional index for each alternate key. Figure 2-4 shows the general structure of an indexed file that

[1] The indexed file organization is supported by the RMS-11K product.

has been defined with only a single key. Figure 2-5 depicts an indexed file defined with two keys -- a primary key and one alternate key.

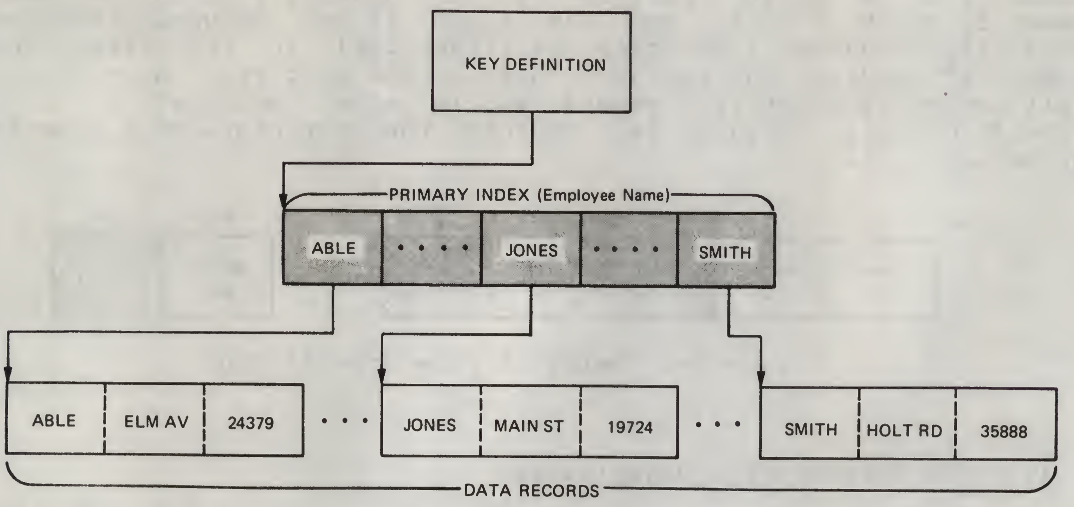


Figure 2-4 Single Key Indexed File Organization

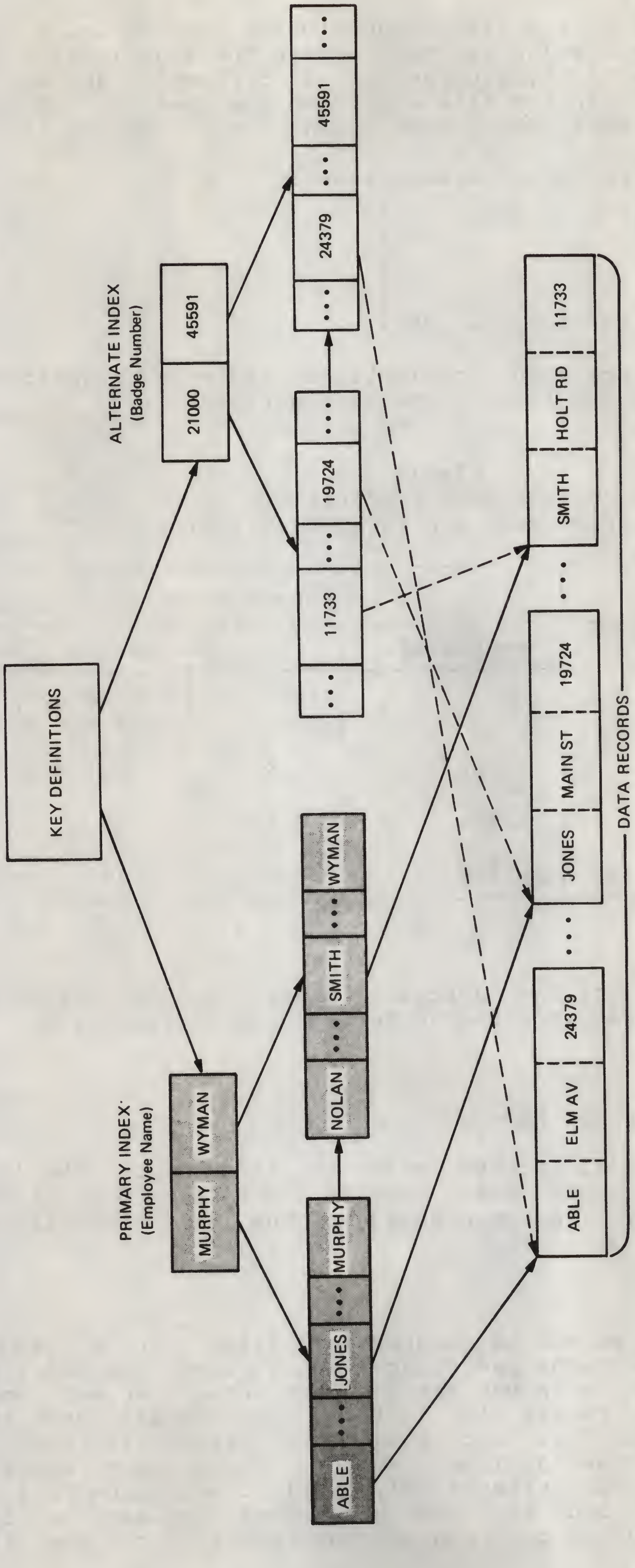


Figure 2-5 Multi-key Indexed File Organization

2.2 RMS-11 ACCESS MODES

The various methods of retrieving and storing records in a file are called access modes. While you must choose the organization of a file at the time you create it, you can use a different access mode to process records within the file each time you open it. Additionally, your program can change access mode during the processing of a file.

RMS-11 provides three record access modes:

1. Sequential

2. Random

3. Record's file address (RFA)

RMS-11 permits only certain combinations of file organization and access mode. Table 2-1 lists these combinations.

Table 2-1
Permissible Combinations of
Access Modes and File Organizations

File Organization	Access Mode			
	Sequential	Random		RFA
		Record #	Key Value	
Sequential	Yes	No	No	Yes[1]
Relative	Yes	Yes	No	Yes
Indexed	Yes	No	Yes	Yes

[1] Disk files only.

The following subsections describe RMS-11 access modes and the capability of changing access mode during program execution.

2.2.1 Sequential Access Mode

You can use sequential access mode to access all RMS-11 files. Sequential access means that records are retrieved or written in a particular sequence. The organization of the file established this sequence.

2.2.1.1 **Sequential Access to Sequential Files** - In a sequentially organized file, physical adjacency establishes the order in which records are retrieved when you use the sequential access mode. To read a particular record in a file, say the fifteenth record, a program must open the file and access the first fourteen records before accessing the desired record. Thus, each record in a sequential file can be retrieved only by first accessing all records that physically precede it. Similarly, once a program has retrieved the fifteenth record, it can read all the remaining records (from the sixteenth on) in physical sequence. It cannot, however, read any preceding record without closing and reopening the file and beginning again with the first record.

When writing new records to a sequential file in sequential access mode, a program must first request that RMS-11 position the file immediately following the last record. Thereafter, each sequential write operation the program issues causes a record to be written following the previous record.

2.2.1.2 **Sequential Access to Relative Files** - During the sequential access of records in the relative file organization, the contents of the record cells in the file establish the order in which a program processes records. RMS-11 has the ability to recognize whether successively numbered record cells are empty or contain records.

When a program issues read requests in sequential access mode for a relative file, RMS-11 ignores empty record cells and searches successive cells for the first one containing a record. If, for example, a relative file contains records only in cells 3, 13, and 47, successive sequential read requests cause RMS-11 to return relative record number 3, then relative record number 13, and finally relative record number 47.

When a program adds new records in sequential access mode to a relative file, the order in which RMS-11 writes the records depends on ascending relative cell numbers. Each write request causes RMS-11 to place a record in the cell whose relative number is one higher than the relative number of the previous request -- as long as that cell does not already contain a record. If the cell already contains a record, RMS-11 rejects the write operation. Thus, RMS-11 allows a program to write new records only into empty cells in the file.

2.2.1.3 **Sequential Access to Indexed Files** - In an indexed file, the presence of one or more indexes permits RMS-11 to determine the order in which to process records in sequential access mode. The entries in an index are arranged in ascending order by key values. Thus, an index represents a logical ordering of the records in the file. If you have defined more than one key for the file, each separate index associated with a key represents a different logical ordering of the records in the file. A program, then, can use the sequential access mode to retrieve records in the order represented by any index.

When reading records in sequential access mode from an indexed file, a program initially specifies a key (e.g., primary key, first alternate key, second alternate key, etc.) to RMS-11. Thereafter, RMS-11 uses the index associated with that specified key to retrieve records in the sequence represented by the entries in the index. Each successive record RMS-11 returns in response to a program read request contains a value in the specified key field that is equal to or greater than that of the previous record returned.

In contrast to a sequential read request, sequential write requests to an indexed file do not require the initial key specification. Rather, RMS-11 uses the stored definition of the primary key field to locate the primary key value in each record to be written to the file. When a program issues a series of sequential write requests, RMS-11 verifies that each successive record contains a key value in the primary key field that is equal to or greater than that of the preceding record.

2.2.2 Random Access Mode

In random access mode, the program, rather than the organization of the file, establishes the order in which records are processed. Each program request for access to a record operates independently of the previous record accessed. Associated with each request in random mode is an identification of the particular record of interest. Successive requests in random mode can identify and access records anywhere in the file.

You cannot use random access mode with sequentially organized files. Both the relative and indexed file organizations, however, permit random access to records. The subsections that follow describe the use of random access with these organizations. Each organization provides a distinct way programs can identify records for access.

2.2.2.1 Random Access in Relative Files - Programs can read or write records in a relative file by specifying relative record numbers. RMS-11 interprets each such number as the corresponding cell in the file. A program, therefore, can read records at random by successively requesting, for example, record number 47, record number 11, record number 31, and so forth. If no record exists in a specified cell, RMS-11 returns a nonexistence indicator to the requesting program. Similarly, a program can store records in a relative file by identifying the cell in the file that a record is to occupy. If a program attempts to write a new record in a cell already containing a record, RMS-11 returns a record-already-exists indicator to the program.

2.2.2.2 Random Access to Indexed Files - The indexed file organization also permits random access of records. However, for indexed files, a key value, rather than a relative record number, identifies the record.

Each program read request in random access mode specifies a key value and the index (e.g., primary index, first alternate index, second alternate index, etc.) RMS-11 must search. When RMS-11 finds the key value in the specified index, it reads the record the index entry points to and passes the record to the user program.

In contrast to read requests, which require a program-specified key value, program requests to write records randomly in an indexed file do not require the separate specification of a key value. All key values (primary and, if any, alternate key values) are in the record itself. When you open an indexed file, RMS-11 retrieves all key definitions stored in the file. Thus, RMS-11 knows the location and length of each key field in a record. Before writing a record into the file, RMS-11 examines the values contained in the key fields and creates new entries in the indexes. In this way, RMS-11 ensures that the record can be retrieved by any of its key values. Thus, the process by which RMS-11 adds new records to the file is precisely the process it uses to construct the original index or indexes.

2.2.3 Record's File Address (RFA) Access Mode

You can use record's file address (RFA) access mode with any file organization as long as the file resides on a disk device. This access mode is further limited to retrieval operations only. Similar to random access mode, however, RFA access allows a specific record to be identified for retrieval.

As the term record's file address indicates, every record within a file has a unique address. The actual format of this address depends on the organization of the file. In all instances, however, only RMS-11 can interpret this format.

The most important feature of RFA access is that the address (RFA) of any record remains constant while the record exists in the file. After every successful read or write operation, RMS-11 returns the RFA of the subject record to your program. Your program can then save this RFA to use again to retrieve the same record. It is not required that this RFA be used only during the current execution of your program. RFAs can be saved and used at any subsequent point in time.

2.2.4 Dynamic Access

Dynamic access is not strictly an access mode. Rather, it is the capability to switch from one access mode to another while processing a file. There is no limitation on the number of times such switching can occur. The only limitation is that the file organization (or, in the case of RFA access, the device containing the file) must support the access mode selected.

As an example, you can use dynamic access effectively immediately following a random or RFA access mode operation. When your program accesses a record in one of these modes, RMS-11 establishes a new current position in the file. Your program can then switch to sequential access mode. By using the randomly accessed record (rather than the beginning of the file) as the starting point, your program can retrieve succeeding records in the sequence established by the file's organization.

2.3 SAMPLE USES OF RMS-11 FILE ORGANIZATIONS

You choose a file organization according to your application's need to access records in a particular way. The following subsections suggest three situations in which a needed access mode determines the selection of the sequential, relative, or indexed file organization.

2.3.1 Sample Sequential File

Consider a particular requirement of a mail-order company as an example of the use of the sequential file organization. Such a company might maintain its master list of customer names and addresses as individual records in a sequential file. This means of organizing information would be efficient for a mass mailing application. A program would access each record in the file in turn and print its contents on a separate mailing label. The program would read the entire file since each record is of equal interest. However, the same file organization would not be efficient if the program needed to access the address of a single individual. In this case, each individual record would have to be read and examined until the correct one was found.

2.3.2 Sample Relative File

A company might choose to maintain product information records in a relative file. The file would be organized so that each cell number (and, therefore, relative record number) would correspond with the unique part number associated with a particular product. This structure would allow easy access to any individual product record in the file if the part number associated with that product were known. Unlike records in a sequential file, there would be no need to access any record in the relative file other than the record of interest.

2.3.3 Sample Indexed File

In creating a file to contain personnel information, a company might select the indexed organization. The employee badge number field might be selected as the primary key. This field, rather than the employee name field, is a suitable primary key because badge numbers within a company are unique, whereas two employees could have the same name. Access to the personnel record of any individual, then, would require only the badge number. The record RMS-11 would return to the application program would contain the specified value in the badge number field. As in the relative file organization, access to a particular record in an indexed file does not require that any other records be accessed first.

CHAPTER 3

FILE ATTRIBUTES

You can use either an application program (written in COBOL-11, BASIC-PLUS-2, or MACRO-11) or the RMS-11 utility called DEFINE (refer to the RMS-11 Utilities User's Guide) to create a file. Each RMS-11 file has certain logical and physical characteristics, known as attributes. Through the source language statements of an application program or the command line invoking the DEFINE utility, you describe these attributes to RMS-11. With this attribute information, RMS-11 begins the structuring of the file on a storage medium.

The most important attribute of any RMS-11 file is its organization. You can tailor a file for use in a particular application by making the proper selection of this and other required or optional attributes. In addition to file organization, you can choose from among the following attributes:

- The storage medium on which the file resides
- The file and protection specification of the file
- The format and size of records
- The size of the file
- The size of a particular storage structure, known as the bucket, within relative and indexed files
- The definition of keys for indexed files.

3.1 STORAGE MEDIA

Your selection of a storage medium on which RMS-11 builds a file is related to the organization of the file. You can create permanent sequential files on disk devices or ANSI magnetic tape volumes. Transient files can be written on devices such as line printers and terminals.

Unlike sequential files, relative and indexed files are restricted to a particular medium. These files can reside only on disk devices.

3.2 FILE AND PROTECTION SPECIFICATIONS

The name you assign to a new file enables RMS-11 to subsequently find the file on the storage medium. You follow the conventions for file specifications of the host operating system. Such conventions may differ among the systems supporting RMS-11.

RMS-11 also allows you to assign a protection specification to a file at the time you create it; again, the format of this specification is the format used by the host operating system. Potential users of the file are classified according to membership in the following classes:

1. System - privileged users as determined by a particular installation (This class is not recognized on RSTS/E systems, where privileged users have unlimited access to files.)
2. Owner - the account under which the file is created
3. Group - users with the same project identifier as the owner
4. World - users in general.

Within each class, you may explicitly elect to deny any or all of the following types of access:

- Read access
- Write access
- Other types of access as determined by the host operating system.

Exact descriptions of membership in class and types of access supported are described in manuals pertaining to the host operating system.

3.3 FORMAT AND SIZE OF RECORDS

When creating a file, you must provide format and maximum size specifications for the records the file will contain. The specified format establishes how each record physically appears in the file on a storage medium. The size specification allows RMS-11 to subsequently verify that records written into the file do not exceed the length specified when you created the file.

3.3.1 RMS-11 Record Formats

RMS-11 supports four record formats:

1. Fixed
2. Variable
3. Variable-with-fixed-control (VFC)
4. Stream

Similar to the selection of a storage medium, the choice of a format for the records of a file depends on a file's organization. Table 3-1 shows the allowed combinations of record format and file organization.

Table 3-1
Record Formats and File Organizations

File Organization	Record Format			
	Fixed	Variable	VFC	Stream
Sequential	Yes	Yes	Yes	disk only
Relative	Yes	Yes	Yes	No
Indexed	Yes	Yes	No	No

3.3.1.1 **Fixed Length Record Format** - The term fixed length record format refers to records of a file that are all equal in size. Each record, then, occupies an identical amount of space in the file.

3.3.1.2 **Variable-Length Record Format** - In variable-length record format, records in a file can be either equal or unequal in length. To allow retrieval of variable-length records from a file, RMS-11 prefixes a count field to each record it writes. The count field describes the length (in bytes) of the record. RMS-11 removes this count field before it passes a record to your program.

RMS-11 produces two types of count fields, depending on the storage medium on which the file resides:

1. Variable-length records in files on disk devices have a 1-word (2-byte) binary count field preceding the data field portion of each record. The specified size excludes the count field.

2. Variable-length records on ANSI magnetic tapes have 4-character decimal count fields preceding the data portion of each record. The specified size includes the count field. In the context of ANSI tapes, this record format is known as D format.

3.3.1.3 **Variable-with-Fixed-Control Record Format** - From your point of view, variable-with-fixed-control (VFC) records consist of two distinct parts, the fixed control area and the user data record. The size of the fixed control area is identical for all records of the file. The contents of each fixed control area are completely under the control of your program and can be used for any purpose. As an example, you might use fixed control areas to store the identifier (e.g., relative record number or RFA) of related records.

The second part of a VFC record is similar to a variable-length record. In other words, it is a user data record, variable in length, and composed of individual data fields.

The two parts of a VFC record correspond to the way your program writes and reads such records. Prior to an output operation, your program builds a VFC record in two locations. It builds the fixed control area in a location separate from the user data part of the record. When writing the record to the file, RMS-11 fetches both the fixed control area and the user data part of the record from their respective program locations. RMS-11 then prefixes the user data part of the record with the fixed control area, prefixes the result with a count field that describes the total size of both parts, and writes the record to the file.

On input operations, RMS-11 reverses the preceding procedure. It uses the count field to locate the entire VFC record in the file. RMS-11 removes this count field. Then, it removes the fixed control area from the record and stores it in one program location while storing the remaining part in a second location.

3.3.1.4 **Stream Format Records** - Records in stream format can be variable in size. However, no count field precedes each record. Instead, RMS-11 considers the entire file a stream of contiguous ASCII characters. Each record in the file is delimited by one of the following:

1. Form feed (FF)

2. Vertical tab (VT)

3. Line feed (LF)

4. Carriage return immediately followed by line feed (CR-LF)

NOTE

> Stream format records are supported for file interchange with non-RMS-11 application programs. Since this format is highly inefficient, it should be used only when such interchange is a concern.

On output operations, RMS-11 examines the last character of the record constructed by a program. If this character is an LF, VT, or FF, RMS-11 leaves the record unaltered and writes it to the file. If the last character is not LF, VT or FF, RMS-11 appends a carriage return (CR) character followed by a line feed (LF) character to the record before writing it to the file.

On input operations, RMS-11 scans the stream of ASCII characters, removing NUL characters and searching for the first occurrence of an FF, VT, LF, or CR-LF combination. If the character that terminates the scan is an FF, VT, or LF (not preceded by CR), RMS-11 passes the entire string, including the terminating character, to the program. If, however, the scan encounters a CR-LF combination, RMS-11 removes these two characters and passes the preceding string as a record to the program. Each successive input operation causes the scan to resume at the character following the last FF, VT, LF, or CR-LF combination encountered.

3.3.2 Size of Records

You must provide RMS-11 with record size information along with the selected record format. RMS-11's use of this information depends on the record format chosen.

When you choose fixed format records, you must indicate the actual size of each record in the file. This size specification becomes part of the information stored and maintained by RMS-11 for the file. Thereafter, if a program attempts to write a record whose length differs from this specified size, RMS-11 will reject the operation.

When creating a file with variable-length format records, you can specify a maximum record size greater than zero or, for sequential and indexed files only, a maximum record size equal to zero. If the specified size is greater than zero, RMS-11 interprets the value as the size of the largest record that can be written into the file. Subsequently, each record that a program actually writes must be less than or equal to this value or RMS-11 rejects the operation. When you specify a record size of zero for a sequential or indexed file, RMS-11 neither checks nor enforces a maximum record size.

VFC format records require two size specifications. The first size specification identifies the length of the fixed control area of all records in the file. The second size specification represents the maximum length of the data portion of the VFC records. RMS-11 handles this second size specification in a manner similar to its handling of the size specification for variable format records. RMS-11 interprets a nonzero value as the size of the largest data portion of a VFC record that a program can write into the file. When, however, this second size specification is zero, RMS-11 neither checks nor enforces a maximum record size.

Finally, for stream format records, RMS-11 permits you to specify the same record size information as for variable format records. That is, a nonzero value represents the maximum permitted size of any record written in the file while a zero value suppresses RMS-11's size checking.

3.4 SIZE OF RMS-11 FILES

The size of an RMS-11 file is expressed as an integral number of virtual blocks. Virtual blocks are physical storage structures. That is, each virtual block in a file is a unit of data whose size depends on the physical medium on which the file resides. For example, the size of virtual blocks in files on disk devices is 512 bytes. Operating system convention has established this size; you cannot alter it. On magnetic tape, a virtual block is the information written between two interrecord gaps. When using ANSI-labeled magnetic tapes, you can specify the size of virtual blocks within files.

The operating system assigns ascending numbers to a file's virtual blocks. This numbering scheme allows a file to appear to you, and to RMS-11 itself, as a series of adjacent virtual blocks. In reality, however, the successive numbering of virtual blocks and the physical placement of these blocks on a storage medium need not necessarily correspond. For example, virtual blocks 167 and 168 in a file, although adjacent in terms of the numbering scheme applied to virtual blocks, need not necessarily occupy adjacent physical locations. Once again, the physical medium on which the file resides determines the handling of virtual blocks.

On magnetic tapes, successively numbered virtual blocks actually occupy successive physical locations. Virtual blocks from one file are never intermixed with virtual blocks from another. On disk devices, however, the situation can be quite different. Files on disk can reside in one or more discrete areas known as extents (see Figure

3-1). Within a single file extent, successively numbered virtual blocks occupy successive physical locations. If a file consists of more than one extent, however, extents of other files or unused space can be interspersed among the extents of the first file.

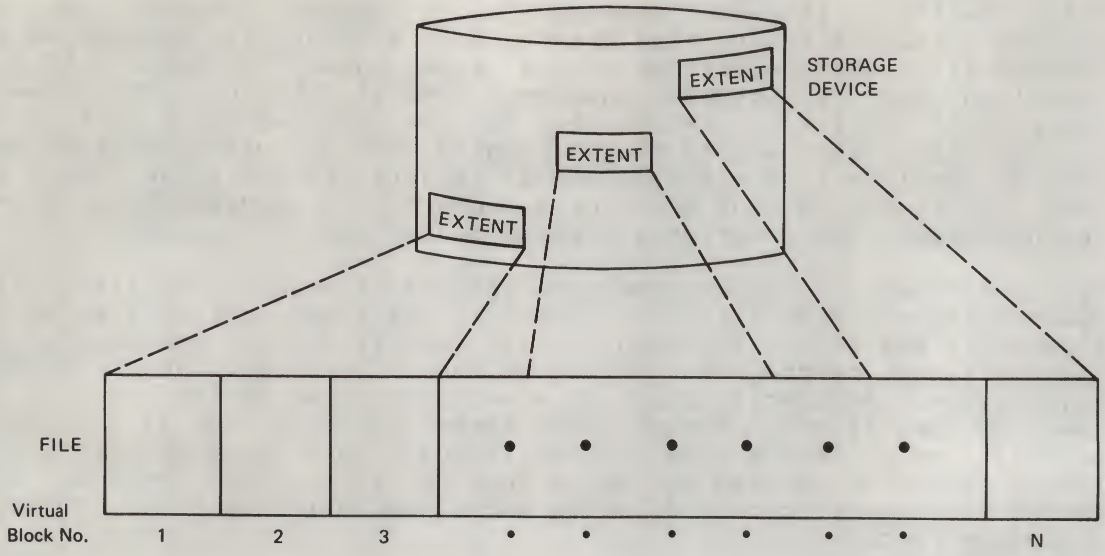


Figure 3-1 Virtual Blocks and Extents

When a file consists of only one extent, it is called a contiguous file since all successive virtual blocks of the file correspond to successive contiguous physical locations. When creating a file, you can request that RMS-11 allocate the file contiguously. The default action is that RMS-11 allocates contiguously as much of the file as possible.

The virtual blocks of a file contain the records your programs write into the file. Depending on the size of records, a virtual block can contain one record, more than one record, or a portion of a record. When a particular record is larger than the size of a virtual block, it is stored in as many successively numbered virtual blocks as needed to contain the entire record. In this instance, the record is said to cross or span block boundaries.

When creating an RMS-11 file, you can specify an initial allocation size. If no file size information is given, RMS-11 allocates the minimum amount of storage needed to contain the defined attributes of the file. If you do specify an initial allocation size, RMS-11 obtains the specified number of virtual blocks for the file. However, this initial size specification does not represent a permanent limit on the size of the file. RMS-11 automatically extends any file whenever the presently allocated space is insufficient to satisfy a program request for storage of data.

3.5 BUCKETS IN RELATIVE AND INDEXED FILES

RMS-11 uses a storage structure known as a bucket for building and maintaining relative and indexed files. Unlike virtual blocks, a bucket can never contain a portion of a record. That is, RMS-11 never permits records to span bucket boundaries.

You define the size of buckets in a file at the time you create the file. Buckets can consist of from 1 to 32[1] virtual blocks. When selecting a bucket size, you must consider file organization, record format, record size, and the internal information RMS-11 maintains in each bucket. Within these constraints, a large bucket size will serve to increase sequential mode processing of a file since fewer actual I/O transfers are required to access records. Minimizing bucket size, on the other hand, means that less I/O buffer space is required to support file processing.

3.6 KEY DEFINITIONS FOR INDEXED FILES

To define a key for an indexed file, you must specify the position and length of character data in the records of the file. You must define at least one key--the primary key--for an indexed file. Additionally, up to 254 alternate keys can be defined. Each such key, primary and alternate, represents from 1 to 255 characters in each record of the file.

When identifying the position and the length of keys to RMS-11, you can define simple or segmented keys. A simple key is a single, contiguous string of characters in the record. In other words, it is a single data field. A segmented key, however, can consist of from two to eight data fields within records. These data fields need not be contiguous. When processing records that contain segmented keys, RMS-11 treats the separate data fields (segments) as a logically contiguous character string.

When defining keys at file creation time, you can specify two characteristics for each key:

1. Duplicate key values are allowed.

2. Key value can change.

When you specify that duplicate key values are allowed, you indicate that more than one record in the file can have the same value in a given key. Such records, therefore, have the same record identifier. The capability to allow duplicate key values further distinguishes indexed files from relative files. In relative files, the record identifier, representing a relative record number, is always unique.

The personnel file can serve as an example of the use of duplicate keys. At file creation time, the creator of the file could define the department name field as an alternate key. As programs wrote records into the file, the alternate index for the department name key field would contain multiple entries for each key value (e.g., PAYROLL, SALES, ADMINISTRATION) since departments are composed of more than one employee. When such duplication occurs, RMS-11 stores the records so that they can be retrieved in first-in/first-out (FIFO) order.

Using the preceding personnel file, an application could be written to list the names of employees in any particular department. A single execution of the application could list the names of all employees working, for example, in the department called SALES. By randomly accessing the file by alternate key and the key value SALES, the application would obtain the first record written into the file containing this value. Then, the application could switch to

[1] The maximum bucket size on the RSTS/E operating system is 15 virtual blocks.

sequential access and successively obtain records with the same value, SALES, in the alternate key field. Part of the logic of the application would be to determine the point at which a sequentially accessed record no longer contained the value SALES in the alternate key field. The program could then switch back to random access mode and access the first record containing a different value (e.g., PAYROLL) in the department name key field.

The second key characteristic (key value can change) indicates that records can be read and then written back into the file with a modified value in the key. When such modification occurs, RMS-11 automatically updates the appropriate index to reflect the new key value. You can specify this characteristic only for alternate keys. Further, when specifying this characteristic, you must also specify that duplicate key values are allowed.

If the sample personnel file were created with the department name field as an alternate key, the creator of the file would need to specify that key values can change. This specification would allow a program to access a record in the file and change the contents of a department name data field to reflect the transfer of an employee from one department to another.

You can also declare the converse of either of these two key characteristics. That is, you can specify for a given key that duplicate key values are not allowed or that key values cannot change. When duplicate key values are not allowed, RMS-11 rejects any program request to write a record containing a value in the key that is already present in another record. Similarly, when the key value cannot change, RMS-11 does not allow a program to write a record back into the file with a modified value in the key.

CHAPTER 4

PROGRAM OPERATIONS ON RMS-11 FILES

After RMS-11 creates a file according to your description of file attributes, your program can access the file and store and retrieve data. When your program accesses the file as a logical structure (i.e., a sequential, relative, or indexed file), it uses access modes to perform record operations that add, retrieve, update, and delete records. The organization of the file determines the types of record operations permitted. If you choose to bypass the record accessing capabilities of RMS-11, your program can access the file as a physical structure. As a physical structure, RMS-11 considers the file simply an array of virtual blocks. To process a file at the physical level, programs use a type of access known as block I/O.

The two sections that follow describe, respectively, record operations and block I/O.

4.1 RECORD OPERATIONS ON RMS-11 FILES

The organization of a file, defined when you create the file, determines the types of operations that your program can perform on records. Depending on file organization, RMS-11 permits your program to perform the following record operations:

- Read a record. RMS-11 returns an existing record within the file to your program.
- Write a record. RMS-11 adds a new record that your program constructs to the file. The new record cannot replace an already existing record.
- Find a record. RMS-11 locates an existing record in the file. It does not return the record to your program, but establishes a new current position in the file.
- Delete a record. RMS-11 removes an existing record from the file.
- Update a record. Your program modifies the contents of a record read from the file. RMS-11 writes the modified record into the file, replacing the old record.

Table 4-1 shows the cominbations of record operations and file organizations that RMS-11 permits. The subsections that follow discuss record operations in the context of each file organization.

Table 4-1
Record Operations and File Organizations

File Organization	Record Operation				
	Read	Write	Find	Delete	Update
Sequential	Yes	Yes	Yes	No	Yes[1]
Relative	Yes	Yes	Yes	Yes	Yes
Indexed	Yes	Yes	Yes	Yes	Yes

[1] Disk files only.

4.1.1 Sequential File Organization Record Operations

In the sequential file organization, your program can read existing records from the file using sequential or RFA access modes. New records can be added only to the end of the file and only through the use of sequential access mode. The find operation is supported in both sequential and RFA access mode. In sequential access mode your program can use a find operation to skip records. In RFA access mode, your program can use the find operation to establish a random starting point in the file for sequential read operations. The sequential file organization does not support the delete operation since the structure of the file requires that records be adjacent in and across virtual blocks. Your program can, however, update existing records in disk files as long as the modification of a record does not alter its size.

4.1.2 Relative File Organization Record Operations

The relative file organization permits your program greater flexibility in performing record operations than the sequential organization. Your program can read existing records from the file using sequential, random, or RFA access mode. New records can be sequentially or randomly written so long as the intended record cell does not already contain a record. Similarly, any access mode can be used to perform a find operation. After a record has been found or read, RMS-11 permits the delete operation. Once a record has been deleted, the record cell is available for a new record. Lastly, your program can update records in the file. If the format of the records is variable, update operations can modify record length up to the maximum size specified when the file was created.

4.1.3 Indexed File Organization Record Operations

The indexed file organization provides the greatest flexibility in performing record operations. Your program can read existing records from the file in sequential, RFA, or random access mode. When reading records in random access mode, your program can choose one of four types of matches that RMS-11 must perform using the program-provided key value. The four types of matches are:

1. Exact key match
2. Approximate key match
3. Generic key match
4. Approximate and generic key match

Exact key match requires that the contents of the key in the record retrieved precisely match the key value specified in the program read operation.

The approximate match facility allows your program to select either of the following two relationships between the key of the record retrieved and the key value specified by your program:

- Equal to or greater than
- Greater than

The advantage of this kind of match is that if the requested key value does not exist in any record of the file, RMS-11 returns the record that contains the next higher key value. This allows your program to retrieve records without knowing an exact key value.

Generic key match means that the program need specify only an initial portion of the key value. RMS-11 returns to your program the first occurrence of a record whose key contains a value beginning with those characters. This capability is useful in applications where a series of records must be retrieved according to the contents of only a part of the key field. In an indexed inventory file, for example, a company might designate its part numbers in such a way that the first three digits represent the vendor from whom the part is purchased. In order to retrieve the record associated with a particular part, the program would normally supply the entire part number. Generic selection permits the retrieval of the first record representing parts purchased from a specific vendor.

The final type of key match combines both the generic and approximate facilities. Your program specifies only an initial portion of the key value, as with generic match. Additionally, your program specifies that the key data field of the record retrieved must be either:

- Equal to or greater than the program-supplied value
- Greater than the program-supplied value

In addition to versatile read operations, RMS-11 allows any number of new records to be written into an indexed file. It rejects a write operation only if the value contained in a key of the record violates a user-defined key characteristic (e.g., duplicate key values not permitted).

The find operation, similar to the read operation, can be performed in sequential, RFA, or random access mode. When finding records in random access mode, your program can specify any one of the four types of key matches provided for read operations.

In addition to read, write, and find operations, your program can delete any record in an indexed file and update any record. The only restriction RMS-11 applies during an update operation is that the contents of the modified record must not violate any user-defined key characteristic (e.g., key values cannot change or duplicate key values not permitted).

4.2 BLOCK I/O

Block I/O allows your program to bypass entirely the record processing capabilities of RMS-11. Rather than performing record operations through the use of supported access modes, your program can process a file as a physical structure consisting solely of virtual blocks.

Using block I/O, your program reads or writes multiple virtual blocks by identifying a starting virtual block number in the file. Regardless of the organization of the file, RMS-11 accesses the identified block or blocks on behalf of your program.

Since RMS-11 files, particularly relative and indexed files, contain internal information meaningful only to RMS-11 itself, DIGITAL does not recommend that you modify a file using block I/O. The presence of the block I/O facility, however, does permit you to create your own file structures. You must, however, maintain the resultant structures using specialized programs. You cannot access these structures through the use of RMS-11 record access modes and record operations.

CHAPTER 5

THE RMS-11 RUNTIME ENVIRONMENT

The environment within which your program processes RMS-11 files at runtime consists of two levels, the file processing level and the record processing level. At the file processing level, RMS-11 and the host operating system provide an environment that permits concurrently executing programs to share access to the same file. RMS-11 ascertains the amount of sharing permissible from information provided by the programs themselves. Additionally, at the file processing level, RMS-11 provides facilities that allow programs to minimize buffer space requirements for file processing. At the record processing level, RMS-11 allows your program to access records in a file through one or more record access streams. Each record access stream represents an independent and simultaneously active series of record operations directed toward the file. Within each stream, your program can perform record operations synchronously or asynchronously. That is, RMS-11 allows your program to choose between receiving control only after a record operation request has been satisfied (synchronous operation) or receiving control possibly before the request has been satisfied (asynchronous operation). Lastly, for both synchronous and asynchronous record operations, RMS-11 provides two record transfer modes, move mode and locate mode. Move mode causes RMS-11 to copy a record from an I/O buffer into a program-provided location. Locate mode allows your program to address records directly in an I/O buffer.

The two sections of this chapter describe, respectively, the file processing and record processing runtime environment.

5.1 THE FILE PROCESSING ENVIRONMENT

RMS-11 provides two major facilities at the file processing level, file sharing and buffer handling.

5.1.1 File Sharing

Timely access to critical files requires that more than one concurrently executing program be allowed to process the same file at the same time. Therefore, RMS-11 allows executing programs to share files rather than requiring them to process files serially. The manner in which a file can be shared depends on the organization of the file. Program provided information further establishes the degree of sharing of a particular file. RMS-11 coordinates the sharing of a relative or indexed file through a bucket locking mechanism. The following paragraphs, therefore, describe:

- File organizations and file sharing
- Program sharing information
- Bucket locking

5.1.1.1 **File Organizations and File Sharing** - With the exception of magnetic tape files, which cannot be shared, every RMS-11 file can be shared by any number of programs that are reading, but not writing, the file. Sequential files on disk can be shared by multiple readers and a single writer. Relative and indexed files, however, can be shared by multiple readers and multiple writers. Your program can read or write records in a relative or indexed file while other programs are similarly reading or writing records in the file. Thus, the information in such files can be changing while programs are accessing them.

5.1.1.2 **Program Sharing Information** - While a file's organization establishes whether it can be shared for reading with a single writer or for multiple readers and writers, your program specifies whether such sharing actually occurs at runtime. You control the sharing of a file through information your program provides RMS-11 when it opens the file. First, your program must declare what operations (e.g., read, write, delete, update) it intends to perform on the file. Second, your program must specify whether other programs can read the file or both read and write the file concurrently with your program.

The combination of these two types of information allows RMS-11 to determine if multiple user programs can access a file at the same time. Whenever your program's sharing information is compatible with the corresponding information another program provides, both programs can access the file concurrently.

5.1.1.3 **Bucket Locking** - RMS-11 uses a bucket locking facility to control operations to a relative or indexed file that is being accessed by one or more writers. The purpose of this facility is to ensure that a program can add, delete, or modify a record in a file without another program simultaneously accessing the same record.

When your program opens an indexed or relative file with the declared intention of writing or updating records, RMS-11 locks any bucket accessed by your program. This locking prevents another program from accessing any record in the bucket until your program releases it. The lock remains in effect until your program accesses another bucket. RMS-11 then unlocks the first bucket and locks the second. The first bucket is then available for access by another concurrently executing program.

5.1.2 Buffer Handling

To your program, record processing under RMS-11 appears as the movement of records directly between a file and the program itself. Transparently to your program, however, RMS-11 reads or writes virtual blocks or buckets of a file into or from internal memory areas known as I/O buffers. Records within these buffers are then made available to your program.

The storage structures transferred between a file and I/O buffers depend on the organization of the file. When your program processes sequential files, RMS-11 reads and writes virtual blocks. For relative and indexed files, RMS-11 reads and writes buckets. Thus, the storage element RMS-11 uses to structure the file is the unit of transfer between the file and memory when RMS-11 accesses the file in response to your program's record operation request.

In addition to buffers that contain virtual blocks or buckets, RMS-11 requires a set of internal control structures to support file processing. The combination of these buffers and control structures is known as the space pool. RMS-11 maintains a separate space pool for each executing program. Rather than allocating space solely on the basis of the total number of files processed, RMS-11 provides facilities to ensure that a space pool is large enough to accommodate only the requirements of the largest number of files that can be open simultaneously. Using these facilities, your program provides information that allows RMS-11 to calculate the minimum size requirements of the space pool.

In providing size requirements for the I/O buffer portion of the space pool, you can choose one of two options:

1. A completely centralized space pool

2. Private I/O buffers for one or more files

In a completely centralized space pool, all I/O buffers as well as the internal control structures required for file processing are inaccessible to your program. RMS-11 totally manages the space within the pool and allocates portions, as needed, as buffer space and control structures for open files.

Unlike a completely centralized space pool, private I/O buffers allow your program some measure of control over I/O buffer space. You can allocate private I/O buffers on a per-file basis by explicitly specifying the address and total size of the buffers to be used for a particular file. While the file is open, RMS-11 manages this buffer space and your program must not access it. However, when the file is closed, the private I/O buffer space is available for use by your program.

The major advantage of private I/O buffers is that they avoid fragmenting a completely centralized space pool. That is, since particular files have varying buffer requirements based on their organization, a centralized space pool could have sufficient space available for the opening of an additional file but the space could be noncontiguous. When such a situation arises, your program can not open the desired file. Such fragmentation cannot occur in a private I/O buffer pool since there is no mixture of differing space requirements.

5.2 THE RECORD PROCESSING ENVIRONMENT

After opening a file, your program can access records in the file through the RMS-11 record processing environment. This environment provides three facilities:

1. Record access streams

2. Synchronous or asynchronous record operations

3. Move or locate record transfer modes

5.2.1 Record Access Streams

In the record processing environment, your program accesses records in a file through a record access stream. A record access stream is a serial sequence of record operation requests. For example, your program can issue a read request for a particular record, receive the record from RMS-11, modify the contents of the record, and then issue an update request that causes RMS-11 to write the record back into the file. The sequence of read and update record operation requests can then be performed for a different record, or other record operations can be performed -- again in a serial fashion. Thus, within a record access stream, there is at most one record being processed at any point in time. However, for relative and indexed files, RMS-11 permits your program to establish multiple record access streams for record operations to the same file. The presence of such multiple record access streams allows your program to process in parallel more than one record of a file. Each stream represents an independent and concurrently active sequence of record operations. Further, when such streams update records in the file, RMS-11 employs the same bucket locking mechanism among streams that it uses to control the sharing of a file among separate programs.

As an example of multiple record access streams, your program could open an indexed file and establish two record access streams to the file. Your program could use one record access stream to access records in the file in random access mode through the primary index. At the same time, your program could use the second record access stream to access records sequentially in the order specified by an alternate index. When your program accesses a record through either stream, RMS-11 automatically uses its bucket locking mechanism to ensure that both streams do not attempt to write the same record at the same time.

5.2.2 Synchronous and Asynchronous Record Operations[1]

Within each record access stream, your program can perform any record operation either synchronously or asynchronously. When a record operation is performed synchronously, RMS-11 returns control to your program only after the record operation request has been satisfied (e.g., a record has been read and passed to your program). When a record operation is performed asynchronously, RMS-11 can return control to your program before the record operation request has been satisfied. Your program, then, can utilize the time required for the physical transfer between the file and memory of the block or bucket containing the record to perform other computations. However, your program cannot issue a second record operation through the same stream until the first record operation has completed. To ascertain when a record operation has actually been performed, your program can issue a wait request and regain control when the record operation is complete.

[1] The RSTS/E operating system supports synchronous record operations only.

5.2.3 Record Transfer Modes

In addition to specifying synchronous or asynchronous operations for each request in a record access stream, your program can utilize either of two record transfer modes to gain access to each record in memory. The following subsections describe these two modes, move mode and locate mode.

5.2.3.1 Move Mode Record Transfers - RMS-11 permits move mode record operations for all file organizations and record operations. Move mode requires that an individual record be copied between the I/O buffer and your program. For read operations, RMS-11 reads a block or bucket into an I/O buffer, finds the desired record within the buffer, and moves the record to a program-specified location.

Before a write or update operation in move mode, your program builds or modifies a record in its own work space. Then, your program issues a write or update record operation request and RMS-11 moves the record to an I/O buffer.

5.2.3.2 Locate Mode Record Transfers - RMS-11 supports locate mode record transfer for read operations to all file organizations. However, it permits locate mode on write operations for sequential files only.

Locate mode reduces the amount of data movement, thereby saving processing time. This mode enables your program to access records directly in an I/O buffer. Therefore, there is normally no need for RMS-11 to copy records from the I/O buffer to your program. To allow the program to access a record in the I/O buffer, RMS-11 provides the program with the address and size of the record in the I/O buffer.

INDEX

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- □ Assembly language programmer
- □ Higher-level language programmer
- □ Occasional programmer (experienced)
- □ User with little programming experience
- □ Student programmer
- □ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________

Street ______________________

City ______________ State ______________ Zip Code or Country ______________

-- Fold Here --

-- Do Not Tear - Fold Here and Staple --

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.